ChinaUnix技术社区、Ubuntu中文论坛鼎力推荐

娓娓道来，妙趣横生，风趣幽默，亲切自然，Linux入门其实很简单



本书内容被数以万计的技术博客和论坛转载过

本书内容在Ubuntu中文论坛上阅读量超过74000次，回帖多达1780余个

清华大学出版社



清华大学出版社
北　京

内 容 简 介

本书是一本与众不同的 Linux 入门读物。作者借鉴历史畅销书《明朝那点事儿》的写作风格，将技术图书以风趣幽默的风格娓娓道来，阅读起来十分过瘾。书中以一个拟人化的 Ubuntu 操作系统为主角，以 Ubuntu 10.04 为基础，讲解了 Ubuntu 系统从安装、配置，到搭建开发平台、投入使用的过程。

本书共 8 章，内容容安排上采取循序渐进的方式，由浅入深地引导读者安装、配置、使用 Ubuntu 系统。其中，第 1 章介绍 Linux 系统的产生和发展概况；第 2 章介绍 Ubuntu 系统的各种安装方法；第 3 章介绍安装系统后的基本设置；第 4 章介绍 Ubuntu 下的常用软件；第 5 章介绍 Windows 下的部分软件在 Linux 中运行的解决方案；第 6 章介绍命令行的使用及脚本的编写；第 7 章介绍 Ubuntu 系统下 C/C++、Java、PHP 开发环境的搭建；第 8 章深入讲解一个软件的编译、调试、打包、发布的过程。

本书适合所有 Linux 入门者和开源软件的爱好者，也适合技术人员作为课外读物学习。对于大、中专院校的学生和培训班的学员，本书不失为一本好教材。

图书在版编目（CIP）数据

Linux 入门很简单 / 刘金鹏等编著. —北京：清华大学出版社，2012.4
ISBN 978-7-302-28098-9

Ⅰ. ①L…　Ⅱ. ①刘…　Ⅲ. ①Linux 操作系统－基本知识　Ⅳ. ①TP316.89

中国版本图书馆 CIP 数据核字（2012）第 030171 号

责任编辑： 冯志强
封面设计：
责任校对： 徐俊伟
责任印制：

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn, http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup. tsinghua. edu. cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 185mm×260mm　**印　　张：** 18　**字　　数：** 450 千字
（附光盘 1 张）
版　　次： 2012 年 4 月第 1 版　**印　　次：** 2012 年 4 月第 1 次印刷
印　　数：
定　　价： 元

产品编号：045149-01

前　　言

——笨兔兔的故事

笨兔兔是什么

Ubuntu 是近年来比较流行的一款开源的 Linux 系统，被广泛应用于嵌入式开发平台、网站架设及家庭用户。Ubuntu 这个词来自非洲南部祖鲁语或豪萨语的 ubuntu 一词，大约是“乐于分享”、“我的存在是因为大家的存在”、“仁道待人”之类的意思，是非洲一种传统的价值观（话说我一直没闹明白非洲人会在什么语境下使用这个词）。这个词的读音大约应该读作[u:`bu:ntu:]，读起来有些拗口，并且 Ubuntu 至今似乎没有正式的中文译名。于是小生不才取其谐音，便称其为“笨兔”，或者再“卖萌”一点，就叫“笨兔兔”了。

Ubuntu 在系统的易用性上较之前的其他发行版有很大的提高。尤其是 wubi 安装方式的引入，使得完全没有经验的用户也可以在不影响已经存在的 Windows 系统的前提下，像安装一个软件一样安装好整个 Ubuntu 系统。这在 Linux 系统的普及和推广方面起了很大作用。对于有兴趣、需要学习 Linux 系统的读者来说，无论你是想要学习 Linux 服务器的架设和维护，还是想要搭建嵌入式开发平台，Ubuntu 都是一个适合的入门系统。

关于本书的由来

小生第一次接触 Linux 是 2001 年左右的事情。第一个被我安装到电脑上的 Linux 系统，是国产的红旗 4.0。不过只是体验了一下，它的启动次数绝对不超过 5 次，之后就被我卸载了。后来陆陆续续又相继安装了 Magic Linux、Mandrake、SuSe、Gentoo、Arch、Ubuntu 等发行版。这里面第一个被我真正用于日常使用的，就是 Gentoo，而使用时间最长的，就是 Ubuntu 了。用的时间长了，经验也就多了些。那阵子常混迹于 Ubuntu 中文论坛，见到一些新手遇到了和我当初经历的类似的问题，就本着“乐于分享”的精神去解答。

一般新手遇到的很多问题是有共性的，有的问题基本上 10 个人里有 9 个都要问一遍。每次都要回答同样的问题有点累，于是我就想写一个帖子，把常见的问题、常用的知识总结在那里。一来是为了给自己的知识做个整理和归纳，二来也是为了偷懒。以后再看到有人遇到了同样的问题，我就可以只说一句：“去看我那某某帖子。”懒得可以吧？

不过虽然有这么个想法，然而一直没有想好怎么写。2008 年年底的时候，恰巧那一阵在看《明朝那些事儿》，该书以戏谑、调侃、机智、幽默的笔法讲述枯燥无味的历史，感觉很有意思。忽一日脑中灵光乍现：历史可以这么写，技术为什么不可以这么写？于是，就有了最初的《笨兔兔的故事》那个连载的帖子。可能我是第一个用讲故事的形式来说技术的吧，值得小小地骄傲一下。

最初的《笨兔兔的故事》是想写成小说式的结构。里面的人物就是一个个拟人化了的软件。剧情呢，就是一个 Ubuntu 系统被一个初识 Linux 的菜鸟安装到计算机上，从此 Ubuntu 系统中的软件们在用户的面前努力表现自己，帮助用户解决日常遇到的各种问题。用户也在使用的过程中从菜鸟成长为不那么菜的鸟（很抱歉我还不算高手，所以写不到炉火纯青那个境界）。这其中穿插着 Linux 系统相关的各种知识的介绍，使得读者可以由浅入深、潜移默化地在看故事的过程中了解到 Linux 系统的知识、精神和精髓（要是您没了解到，那是我没写好）。这篇《笨兔兔的故事》从 2008 年 11 月开始连载，前后历时将近 3 年，共 150 篇。虽然效果还算可以，不过我的第二个偷懒的目的，终于没能达到。因为写成了一个小说式的故事，技术上的内容安排得比较凌乱。再遇到有新人发问，我虽然能够记起这个问题在我的《笨兔兔的故事》里有解释，但我如果只是说“去看《笨兔兔的故事》”，那么人家不免追问“150 回呢，看哪段呀？”说实话，我也记不清在哪段……

于是，就有了写本书的想法。想要把“故事”写得更“技术”一点。在内容的编排上，以 Ubuntu 10.04 为基础，以系统的安装、配置、应用的过程为主线。在讲述故事的同时指导新手安装和配置 Ubuntu 系统。内容与原版那个帖子有少部分重复，是原版故事的重新编排和修改，并添加了新的、更深入的内容。本书旨在增强故事对新手的指导作用，提高实用性，同时，保持一定的故事性，让本书看起来更加生动有趣。

本书的显著特色

- 以讲故事的形式来讲述技术。相信阅读本书的时候不会枯燥无味，能不能做到引人入胜，要看读者的评价了。
- 另外，这本书的视角独特。它是以一个拟人化的 Ubuntu 系统的口吻，以第一人称的形式，讲述发生在计算机里的故事。用一句话概括就是：笨兔兔讲述自己的故事。目前，其他关于 Linux 的技术性书籍中，还没有从如此角度来编写的。
- 再有，就是书中大量的形象生动的比喻和对比。通过将 Linux 系统中的一些概念与生活中常见的事物作对比，帮助读者更好地了解、更深刻地记忆相关的知识。

适合什么样的读者

- Linux 入门新手；
- PHP、Java、C/C++开发人员；
- Linux 技术爱好者；
- 网络管理员和网络维护人员；
- 开源软件爱好者；
- 嵌入式开发初级人员。

本书作者与致谢

本书由刘金鹏主笔编写。其他参与编写和资料整理的人员有陈杰、陈冠军、项宇峰、张帆、陈刚、程彩红、毛红娟、聂庆亮、王志娟、武文娟、颜盟盟、姚志娟、尹继平、张

昆、张薛。

从 2008 年到现在，很多人给予了我很大的帮助。在此，要感谢 Ubuntu 中文论坛的 adagio、yexiaoxing、byd123、ubuntu1023、UWLinux、Hello World!、tenzu、JiangHui、月下叹逍遥、ljj_jjl2008、hceasy、黄美姬、peteryeh64、速腾 1994、nmsfan、wangdu2002、eexpress 及很多我暂时记不起名字的网友的大力支持。感谢他们提出的每一个建议、提供的每一张截图、顶的每一层楼、灌的每一滴水。

同时也感谢我的妻子及家人对编写工作的支持，感谢他们为我做的每一顿饭，帮我找到的每一个错字，排除的每一处歧义。

编著者

目　　录

第 1 章　一切的起因

我们将要讲述的故事，发生在一个你可能不熟悉、不了解，却又几乎天天都会接触到的世界。那里可闻鸟语却无花香，那里不见天日却色彩斑斓——那就是软件的世界。而我们的主角，就是一个软件，一个操作系统——Ubuntu。不过主角总是不着急登场的，在他露面之前，我们先来说说这位主角，以及跟他有血缘关系的几个操作系统的诞生过程。

1.1　UNIX 的诞生

UNIX 是一个强大的操作系统，稳定程度令人发指。要说 UNIX，那得从 20 世纪 60 年代的美国开始说起。

1.1.1　操作系统的从无到有

20 世纪 60 年代，计算机可是个新鲜玩意儿，没有现在这么普及，而且非常笨重，都是用在商业或者科学研究领域，家庭用户想都不用想。而且那时候的计算机远没有易用到 70 多岁的赵大妈都能拿它来斗地主的地步。那时候使用计算机需要由专业人员通过输入一条条的指令，来进行各种运算。他们输入的指令大约相当于现在的汇编指令，所以效率和操作难度有多高就可想而知了。那时候计算机大都没有什么操作系统，顶多有个批处理系统，可以把要输入的指令记录在某种媒介上（比如纸带，如图 1.1 所示）一次性输入进去，省去了重复输入指令的麻烦。

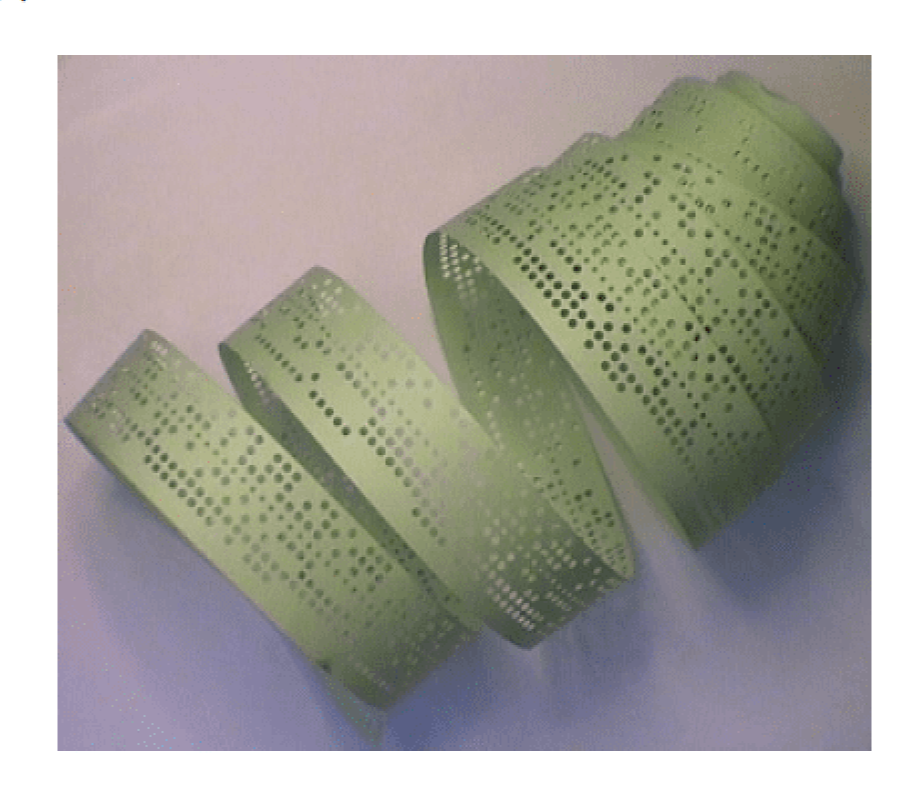


图 1.1　打孔纸带

后来慢慢有了很简单的操作系统，但并不像现在我们见到的操作系统这样通用。这个时候，卖计算机的厂商要为每个型号的计算机设计不同的操作系统，一个程序在这个型号

的计算机上写好了，如果想拿到其他型号的计算机上运行，就需要再重新写一遍，因为这两台机器的硬件组成、操作系统等都不一样。

计算机要是就这样下去，那么 70 多岁的赵大妈就别想玩斗地主了。所幸这个斗地主的问题，后来被那个时代 IT 业界的大地主，蓝色的 IBM 公司率先着手解决了。1964 年 IBM 公司推出了一个系列的大型机，用途、价位各不一样，但它们上面运行的操作系统，都是 System/360，图 1.2 所示就是其中一个。IBM 的这一举措为其带来了很大的利润，因为省去了为每一台电脑单独编写系统的成本。直到今天，IBM 的大型机上依然可以运行这个 360 系统，可见其当初设计时充分考虑了兼容性。然而我们要讲的主角不是 360，而是另一个伟大的操作系统。

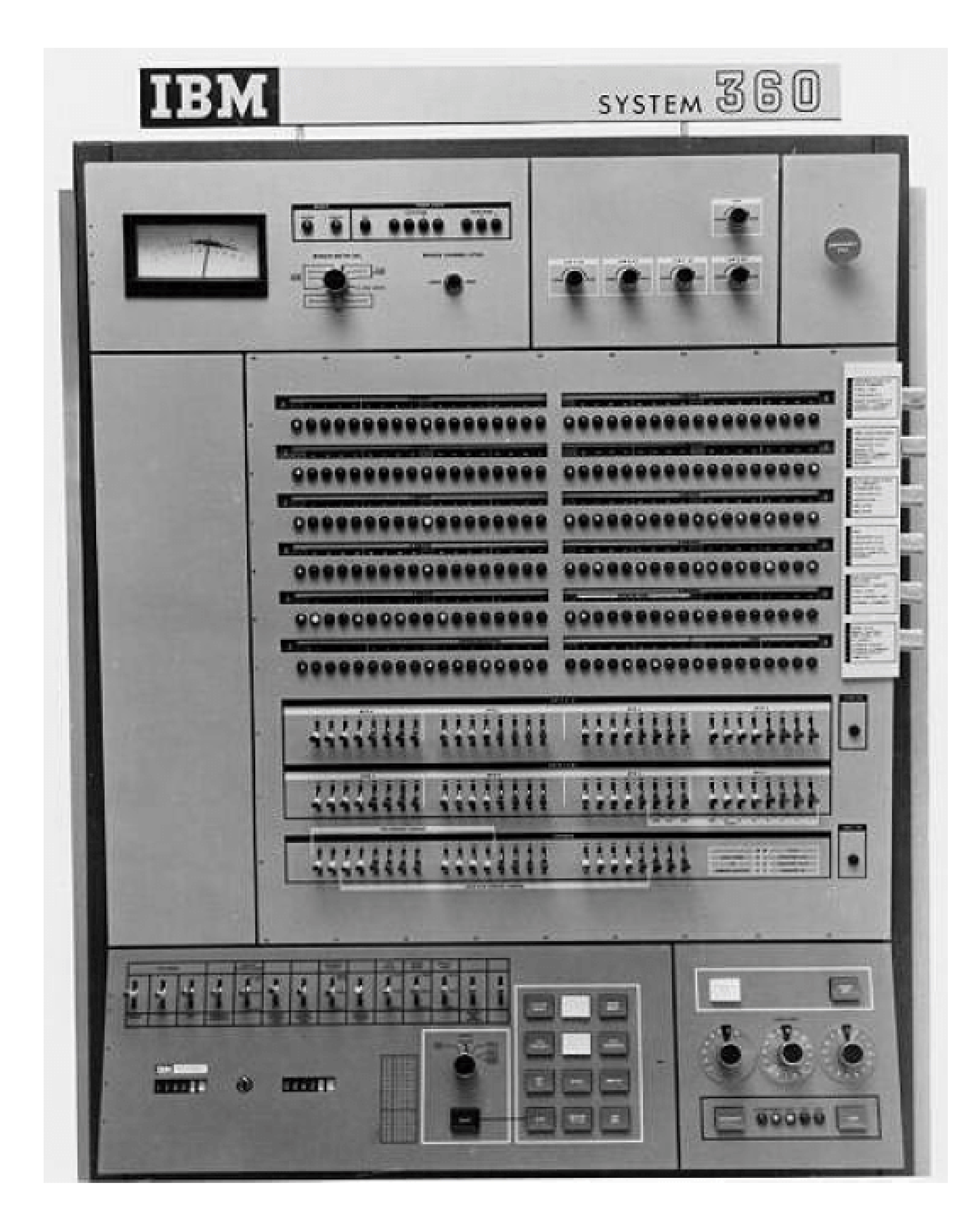


图 1.2　IBM 的 System/360 大型电脑

1.1.2　有牛人的地方就有新技术

【在那牛人聚集的地方】

那时候有个聚集了很多牛人的地方，叫做贝尔实验室，是 1925 年由 AT&T 公司成立的，就是图 1.3 所示的这个地方。一帮头脑发达四肢也不一定简单的家伙整天聚在那里，研究新奇的东西，什么任意门啊，竹蜻蜓啊……都不是他们发明的（听说发明这些的人是个日本科学家）。贝尔实验室那帮人的研究工作大致可以分为 3 个类别：基础研究、系统工程、应用开发。在基础研究方面主要从事电信技术的基础理论研究，包括数学、物理学、材料科学、计算机编程理论等，反正都是大学时听不懂的那几门就是了。系统工程主要研究构成电信网络的高度复杂系统。应用开发部门是贝尔实验室最大的部门，负责设计构成贝尔系统电信网络的设备和软件。具体来说，贝尔实验室研究出来过的东西有晶体管、发光二极管、通信卫星、电子数字计算机、蜂窝移动通信等，都是我们今天的生活中非常

用的东西。总之，通信网的许多重大发明都诞生自这里。



图 1.3　贝尔实验室总部

那时候还有个聚集了很多牛人的地方，叫做麻省理工学院（MIT）。这是美国的一所综合性私立大学，有“世界理工大学之最”的美名，图 1.4 所示是它的主校区。从这里走出的牛人很多，到 2009 年为止，先后有 76 位诺贝尔奖得主，都曾经在麻省理工学院学习或者工作。麻省理工学院的自然及工程科学在世界上享有极佳的盛誉，其管理学、经济学、哲学、政治学、语言学也同样优秀。另外，麻省理工学院研发高科技武器和美国最高机密的林肯实验室、领先世界一流的计算机科学及人工智能实验室、世界尖端的媒体实验室和培养了许多全球顶尖首席执行官的斯隆管理学院，也都是麻省理工学院赫赫有名的宝贵资产。



图 1.4　麻省理工学院主校区

那时候，又有个聚集了很多牛人的地方（哪来这么多地方阿！）。这个地方是个公司，叫做通用电气。这个公司当年是个卖灯泡的，他们的灯泡虽然不节能，寿命也不长，价格还挺贵，但是他们的灯泡非同一般——他们是第一家卖灯泡的！他们的老大，就是大名鼎鼎的托马斯·爱迪生。1876 年，发明灯泡的爱迪生同学成立了爱迪生灯泡厂，为节约蜡烛和灯油作出了突出的贡献。到 1890 年，爱迪生同学将灯泡厂重组，成立了爱迪生通用电气公司，到 1892 年又与汤姆森－休斯顿电气公司合并，成立了通用电气公司。现在，通用公司的总部位于纽约市的通用电气大厦，就是图 1.5 中所示的这座建筑。

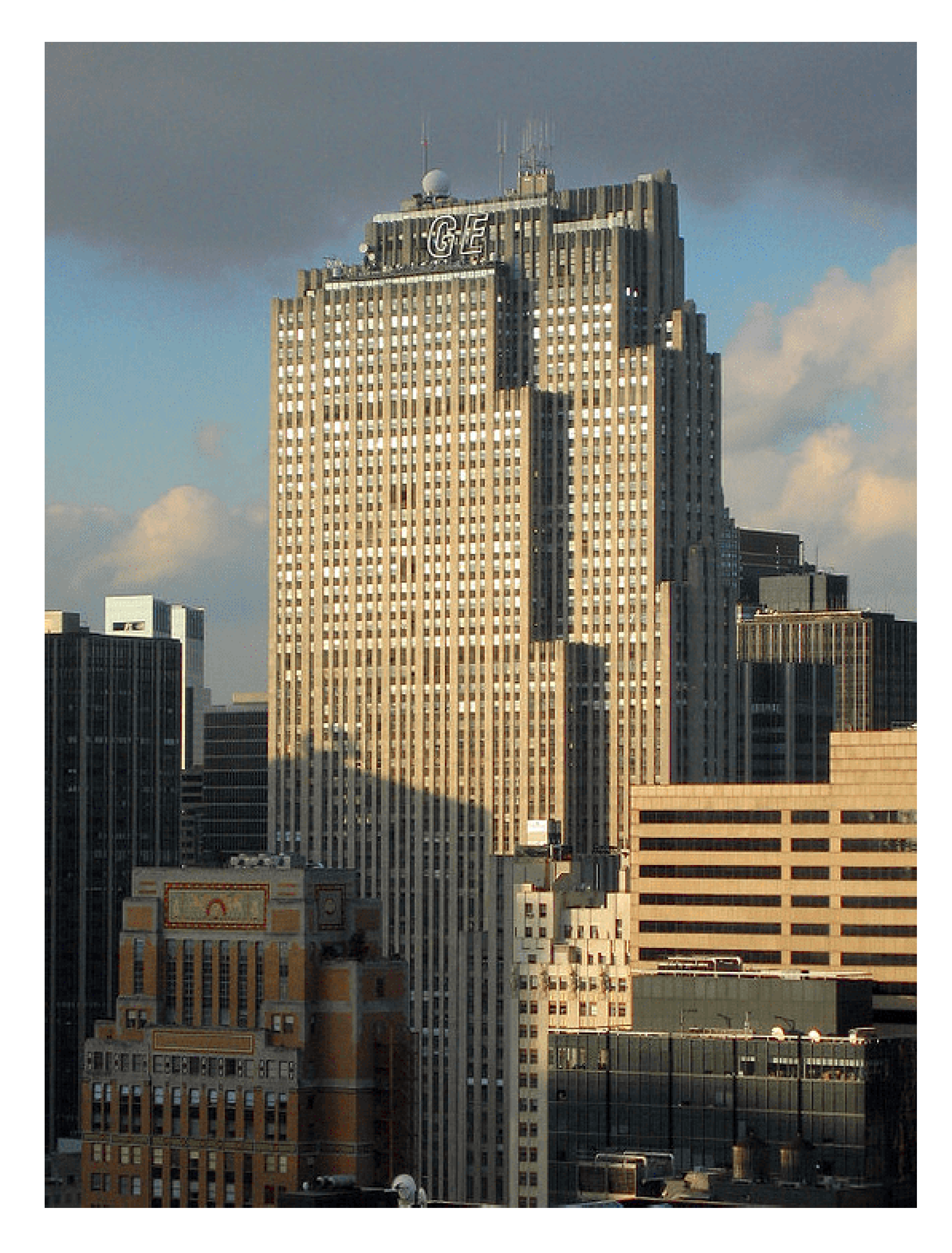


图 1.5　位于纽约市的通用电气大厦

【牛人多了也不一定靠谱儿】

好，时间到了 1965 年，这 3 个聚集着不少牛人的地方有一天忽然想合作一把。他们一起开始了一个制作操作系统的计划。为了结束长期以来计算机上面没有统一的操作系统的混乱局面，他们决定，要创造出一套旷古烁今、空前绝后、惊世骇俗的操作系统！具体来说，这个操作系统应该是一个支持多使用者、多任务、多层次的操作系统。因为这三多，所以这个操作系统就起名叫做 MULTICS——就是 MULTiplexed Information and Computing System 的缩写，连图标都设计出来了，就是图 1.6 所示的这样。有了这 3 家的强强联合，那开发的结果还用问吗？这个 MULTICS 操作系统的项目在 1965 年成立，到了 1969 年就……被取消了。原因好像是因为进度太慢，看来编写操作系统也不是一件容易的事儿嘛。毕竟道路是曲折的，研究是辛苦的，成绩还是有的，失败呢……也是可以原谅的嘛。

项目失败了，大家都很沮丧。在这些沮丧的人中，汤普逊（Kenneth Lane Thompson）只是很普通的一个，就是图 1.7 中所示的这位。汤普逊于 1943 年出生在美国新奥尔良市。在烤翅的芳香中长大的他，没有辜负养育他的父母和那些没有了翅膀的鸡。1960 年，他考

上了加州大学博克莱分校主修电气工程，顺利取得了电子工程硕士的学位。1966 年，他加入了贝尔实验室，参与了 MULTICS 项目。做项目是个很辛苦的事情，在疲劳地揉揉因熬夜而发红的眼睛后，他很想能有个电脑游戏来玩玩。然而那时候别说超级玛丽，连吃豆也没有。所以汤普逊同学就自己编了一个游戏，叫做星际旅行。

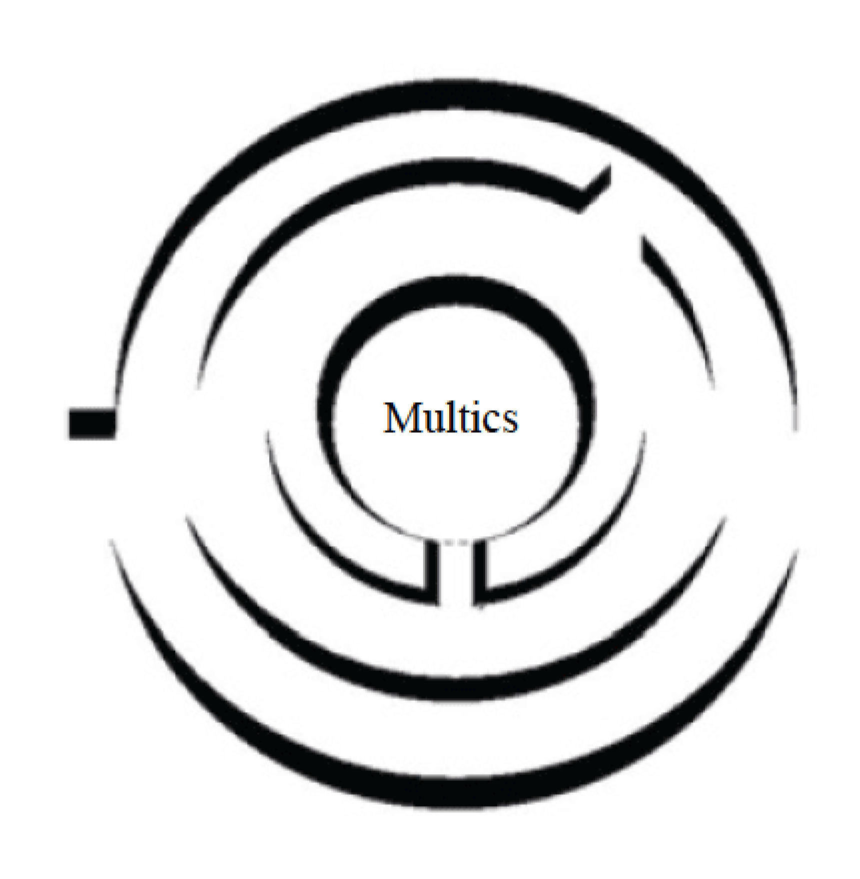


图 1.6　MULTICS 的图标

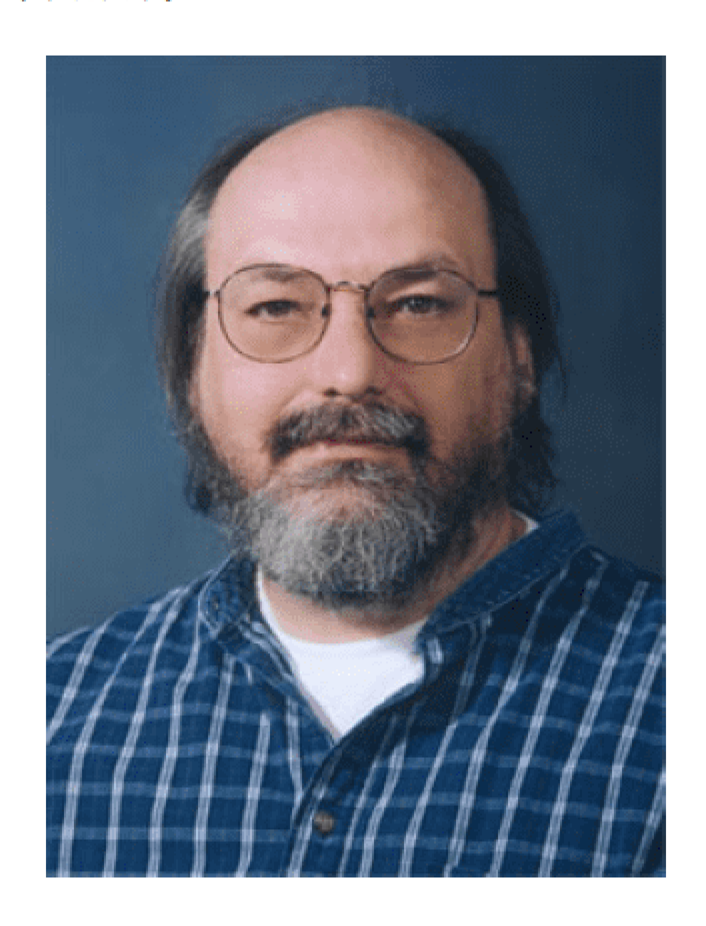


图 1.7　Kenneth Lane Thompson

1.1.3　一个游戏引发的变革——UNIX 元年来了

这个星际旅行跟星际争霸肯定是没得比的，不过在那时候已经算很有吸引力了。这个游戏自然是被设计运行在 MULTICS 系统上的，由于 MULTICS 系统还不完善，导致游戏运行得不是很流畅，所以，能够顺畅地玩星际旅行，成为汤普逊同学努力工作的源动力。

可是后来项目取消了，汤普逊同学似乎再也不可能流畅地玩他的星际旅行了，这是多么遗憾的事情啊。然而汤普逊同学没有就此放弃，强烈的游戏欲望推动他行动起来。毛主席教导我们：自己动手，丰衣足食。我估计汤普逊没有背过毛主席语录，但是他用自己的行动证明了这句话的正确性。他在墙角淘换出一台 PDP－7 的机器，并且伙同其同事 Dernis Ritchie，打算将星际旅行移植到这台 PDP－7 上，于是这台 PDP—7 也跟着名垂史册，就是图 1.8 所示的这台。

当然，要想运行这个游戏，肯定还得有个系统。可是 MULTICS 已经停工了，并且这个系统绝对不是两个人可以搞定的（那么多牛人一起都没搞定）。那怎么办？那就自己动手！于是汤普逊和 Dernis Ritchie 发扬自己动手的精神，用汇编语言写出一个系统，这就是最初的、简陋的、UNIX 的前身。这个系统不像 MULTICS 那么牛，不支持很多的用户，只能支持两个用户（就是他俩玩嘛），支持的进程也有限，其他功能也都没有 MULTICS 设计得那么复杂。相对于那个 MULTICS 系统，Brian Kernighan 戏称他们的系统其实是：“UNiplexed Information and Computing System”，缩写为“UNICS”。后来大家取其谐音，就诞生了 UNIX 这个词。这一年，已经是 1970 年，史称 UNIX 元年。直到现在，计算机中都是用 1970 年 1 月 1 日 0 点 0 分 0 秒作为记录时间的原点。

提示： 计算机中记录的时间，是自 1970 年 1 月 1 日 0 点 0 分 0 秒开始，到当前时间所经历的总秒数，再根据这个秒数计算出具体的年、月、日、小时、分等信息。

后来，Dernis Ritchie 觉得用汇编写的系统不好维护，于是……他也发扬自己动手的精神，发明了 C 语言（符合大牛一切自己动手的风格），然后用 C 语言把 UNIX 重写了一遍。从此，UNIX 走上了发展的快车道。如今，许多世界级的大服务器，用的依旧是 UNIX 系统。

而这一切的努力，最初都是为了玩个游戏。



图 1.8　PDP—7

1.2　Stallman 和他的 GNU 计划

这回要说的，是另一个传奇人物——Richard Matthew Stallman，就是图 1.9 里这位不爱刮胡子的大叔。

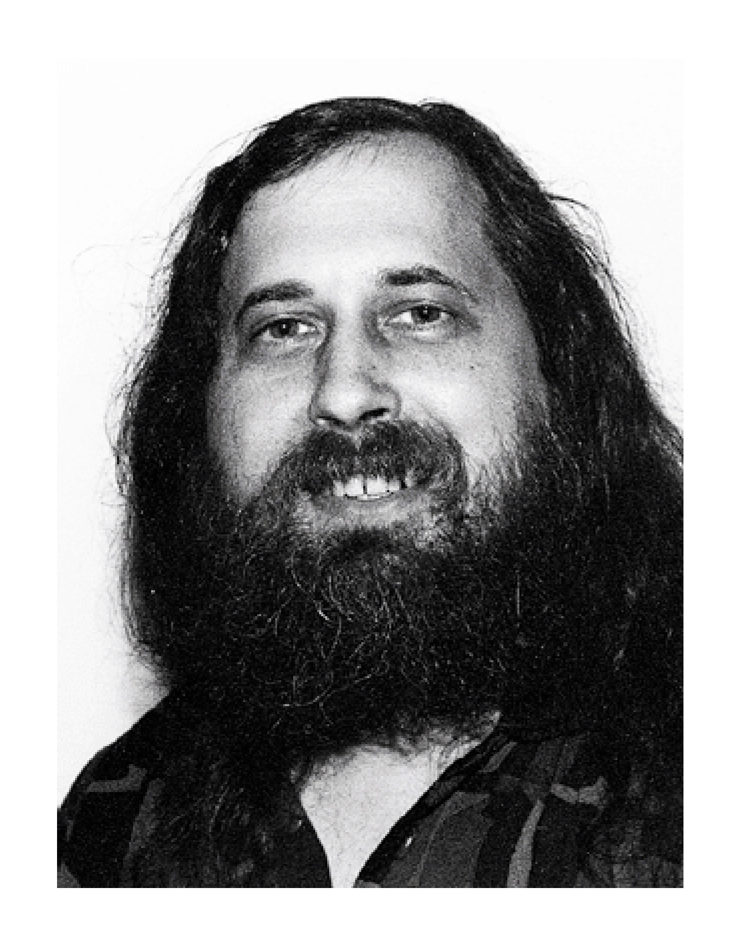


图 1.9　Richard Matthew Stallman

Richard Matthew Stallman，1953 年出生在美国纽约曼哈顿地区。在他生命的前十几年

中，他并没有表现出什么过人的地方，但那是因为他没遇到一个叫做电脑的东西。

1.2.1　快乐的自由

高中的一个暑假，他去给 IBM 打工，花了两周的时间用 Fortran 语言编了一个数据处理的程序。这是他第一次接触计算机，或许就是这次相遇，确定了他未来行走的方向。1971 年，他考上了哈佛大学，上学的同时，他还受聘于麻省理工学院的人工智能实验室，成为了一名职业黑客（黑客这个词没有贬义）。在人工智能实验室期间，他可没少干活，开发了很多有用的软件，其中最著名的就是 Emacs 编辑器。Emacs 是一个可与 Vi 相抗衡的强大的编辑器。两者的操作方式完全不同，但同样强大，各自用自己独有的方式，提高着人们的编辑效率。直到今天，仍然有人争论到底 Emacs 好还是 Vi 好，信奉 Emacs 的人和信奉 Vi 的人形成了两个帮派，这两个帮派经常在互联网上用鼠标键盘相互灌水拍砖，拼个你死我活。哦，扯远了，咱还回来说 Stallman。

那时候的 Stallman 在人工智能实验室里工作得非常愉快，大家有 BUG 同当，有代码共享。那时候的软件工程师的世界，是一个“人人为我，我为人人”的理想世界。因为最初的计算机软件没有什么开源不开源的概念，那时候的软件天生就是自由的！卖计算机的同时会附带软件，包括软件的源代码和文档。计算机厂商卖的主要是计算机的硬件，软件只是附属品而已。用户可以根据自己的需要去修改软件，与别人分享软件。总之，软件是用户花钱买硬件时附带着买来的，用户想怎么玩就怎么玩。软件开发者的目的，也不是靠软件赚钱，而是靠软件支撑起硬件的功能，然后靠卖硬件赚钱。

1.2.2　自由逐渐远去

然而随着技术的发展，软件逐渐脱离硬件成为一个独立的产业，很多软件慢慢地只提供二进制代码而不提供源代码了，这就意味着你不能修改它，并且多数软件还规定最终用户没有二次分发的权利。也就是说，这东西你买了，只能你用，你再给别人就不行！这就好像我买了把菜刀，然后卖菜刀的告诉我“你这把菜刀不许借给你的邻居用，也不许私自给菜刀换刀把，否则我就告你！”

Stallman 当时就遇到了类似这样的菜刀问题。那时候，他们实验室买的第一台打印机附带有驱动程序的源代码。他们那的黑客们可以随意修改这个驱动，根据自己的需要添加些小功能，改改 BUG 之类的，这为他们的工作带来了很大的方便。后来，实验室又买了一台激光打印机，这次厂商只提供了二进制的打印机驱动程序，它是实验室里仅有的一个没有源代码的软件。Stallman 很不喜欢这样的产品，然而他没有选择，只能沉默。

后来出于工作的需要，Stallman 想修改一下这个驱动程序，但是不行，没源代码啊。Stallman 听说卡内基•梅隆大学有这个打印机的驱动程序源代码，他就去了那里，跟他们套近乎：“那啥，大家都是道上混的，谁还没个马高蹬短的时候？是兄弟的拉哥们儿一把，我也没啥事儿，就是我们那打印机老丢字，老把一些关键的字打成口口，我估计是驱动的问题，听说你们这有这驱动的源代码，能不能给我拷一份？”对方办事效率还是挺高的，很干脆地拒绝了他。因为他们和厂商签署了一份保密协议，协议要求他们不能向别人拷贝源代码。Stallman 顿时感到他们背叛了自由的计算机社团，他非常生气，但是他没有办法

改变什么，只好又选择了沉默。

这只是一件小事，只是一个时代的缩影。那个时代，正处在软件向私有化转变的过程中，也是软件逐渐商业化的过程。越来越多的软件选择了不开放源代码，不允许二次分发的发布方式。Stallman 身边的同事，一个一个地跑到开发私有软件的公司去打工了，他们不再相互分享，不再相互交流。Stallman 问：“你们那软件的查找算法做得不错啊，怎么实现的？”“对不起，无可奉告。”“你们的文档工具效率挺高啊。”“对不起，商业机密。”……面对这一切，Stallman 又能说什么呢？他还是只有沉默。

1.2.3　不在沉默中爆发，就在沉默中灭亡

Stallman 爆发了！他不能容忍软件世界里清新自由的空气被私有软件污染；他不能容忍被剥夺按照自己的需求修改软件的权利和乐趣；他不能容忍自己买条皮带尺寸不够时，自己竟然连在上面多打个洞的权利都没有！于是，他就爆发了。

他要重现当年那“人人为我，我为人人”的合作互助的软件世界；他要把使用、复制、研究、修改、分发软件的权利还给软件世界的每一个人民；他要用自己的行动告诉人们，软件天生就该是自由的！

他要开辟一个新的世界，哪怕是一个人在战斗！于是，一个宏伟的计划——GNU 计划在他心中产生了。它的目标是创建一套完全自由的操作系统。因为操作系统是电脑中最重要、最基础的软件，要创造自由的软件世界，自然先要有一套自由的操作系统，然后再以此系统为中心，开发各种各样自由的软件。1983 年，Stallman 在 net.unix-wizards 新闻组上公布了 GNU 计划，这个计划的标志是一头角马（也就是非洲牛羚），就是图 1.10 所示的这个。



图 1.10　GNU 计划的图标

提示： GNU 是“GNU is Not UNIX”的递归缩写，Stallman 表示这个词应该读作/'gnu:/（发音类似“革奴”），以区别于表示非洲牛羚的单词 gnu（发音与“new”相同）。

这个计划要创造一套自由的类 UNIX 操作系统。系统本身及系统上的软件都是自由软件，它们可以被免费获取，随意使用、修改和再分发。并且每个人都可以获得这个系统全部的源代码，每个人都可以为完善这个系统作出自己的贡献。这个系统要使用与 UNIX 相同的接口标准，这样，就可以由不同的人，分期分批地创作操作系统的不同部分而不必担心相互之间协同工作的问题。

1.2.4　实现 GNU 梦想

为了实施 GNU 计划，1985 年，Stallman 又创建了自由软件基金会。基金会的主要工作就是执行 GNU 计划，开发更多的自由软件。1989 年，Stallman 与基金会的一群律师们起草了广为使用的《GNU 通用公共协议证书》也就是 GPL 协议，以此协议来保证 GNU 计划中所有软件的自由性。到了 1990 年，GNU 计划中的这个系统已经初具规模，有了很多优秀的软件。其中有很多是世界各地的黑客们无偿提供的，也有一部分是利用自由软件基金会的基金雇用程序员来开发的，当然，Stallman 自己也身先士卒，开发了 Emacs、GCC、GDB 等重要软件。当他看着这些丰富的自由软件的时候，感觉到那清新自由的空气，终于又回来了，以后，人们就可以拥有一个可以自由使用、自由修改、自由分发的、自由的操作系统了！不过等一下，好像还差点什么，哦，还……差个内核吧。

作为一个系统，没有内核是不行的，这么重要的部件 Stallman 当然不会忘记，所以才会有 Hurd 内核。这个内核被设计为一个遵守 POSIX 标准的微内核。所谓微内核，是相对于宏内核来说的。宏内核就像我们现在的 Linux 内核，是一个独立的程序，里面包含了进程管理、内存管理、文件管理等功能。而微内核则将一个内核需要的功能尽量地简化并且拆分，运行起来是几个独立的程序，有的专门负责进程管理，有的专门负责内存分配。内核是一个系统的核心，所以至关重要，Stallman 对 Hurd 的开发也是精益求精，非常谨慎，以至于内核的进度有些落后于其他的系统软件，当其他软件都已经有比较优秀的版本的时候，Hurd 内核依然不能够走出实验室投入真正的使用。这种情况一直持续到 1991 年，另一位英雄的出现——不过，这里先卖个关子，暂且不去说他。

无论怎样，到今天，Stallman 理想中的自由世界，终于拉开了那沉重的幕布，展现出了自由的光彩。而 Stallman 并不满足，也确实没有满足的理由，这个自由的世界还需要成长，还需要更加丰富多彩，还需要有更多的人走进这个世界中来。于是 Stallman 奔走于世界各地，告诉人们有这么一个自由的世界，号召人们加入这个世界，鼓励人们为使这个世界更加自由而付出自己的力量。他是一个执着的苦行僧，为了他的梦想，为了他的自由世界，他会一直走下去……

1.3　从异想天开到 Ubuntu

1988 年，芬兰赫尔辛基大学迎来了一位新的大学生——Linus Benedict Torvalds，就是图 1.11 所示的这位。当然，那时候他还比较瘦，而且他的名字在学校的花名册中也并不显眼，但是一年后，他大二的时候，开始有故事了。

1.3.1　Minix

大学二年级的时候，Linus 同学开始学习操作系统这门课程。那时候这门课程使用 Minix 系统进行教学。Minix 这个名字或许您听着并不熟悉，这是个专门用于教学的操作系统，

它的系统结构和 UNIX 系统是类似的。有人可能会问：那为什么不直接用 UNIX 呢？嗯，UNIX 确实很先进，很优秀，确实值得学习计算机科学和操作系统的同学们学习。然而要知道有一种东西叫做版权，即便你不怎么在乎这个东西，但人家学校是不能做违法的事的。UNIX 并不免费，并且是天价的，广大穷苦的大学生们买不起，学校也没钱为每一名学生配备一套 UNIX 系统。

荷兰阿姆斯特丹 Vrije 大学的 Andrew S. Tanenbaum 教授（就是图 1.12 所示的这位）在教学过程中就深刻地体会到，世界上缺少一个教学用的操作系统。他的学生们学习了计算机，学习了操作系统原理，不能光啃书本，总得实践一下吧？总得找台机器装个操作系统用用吧？用什么操作系统来教学呢？买个 DOS 装上？虽然那时候 DOS 已经问世了，但是这么一个单用户、单任务、效率也不高的操作系统，实在不能指望它培养出什么软件人才。装个 UNIX？学校还不想破产。于是牛人 Andrew S. Tanenbaum 拿起键盘——咱自个儿编一个吧！然后 Minix 就诞生了。



图 1.11　Linus Benedict Torvalds



图 1.12　Andrew S. Tanenbaum 教授

Minix 取 Mini UNIX 之意，1987 年被编写出来，到 1991 年发展到 1.5 版，后来发展到 2.0 基本上就停止了。因为这个操作系统的初衷只是作为一个教学模型，并不是一个实用的系统，所以功能很简单，体积也很小，并且以后也没有进行进一步的开发和扩充。它为的是能够让学生在一学期内学完整个系统。很长一段时间后又出了 Minix 3，这回有图形界面了，图 1.13 所示就是 Minix 3 的界面。那时候 Minix 在大学中用于教学是免费的，但是用于其他用途是需要给钱的，不过现在已经彻底免费了。它作为一个操作系统，其实并不算优秀，但它是一个源代码完全开放的操作系统，这使得有理想、有志向、有抱负的黑客们，第一次能够完整地阅读到一个操作系统的全部代码。

1.3.2　异想天开的 FREAX

Linus 他们学校的计算机上装的就是这个专门用于教学的 Minix 系统。虽然适合拿来学习，不过系统本身并不强大。这要是别人也还罢了，可是 Linus 同学有个最大的爱好，就是虐待计算机。他热衷于测试计算机的能力和限制，整天研究怎么让计算机按照自己的想

法去干活，怎么发挥计算机最大的性能，一定要把可怜的机器累得精疲力尽，口眼歪斜，电容爆浆，吐血身亡才算罢休。

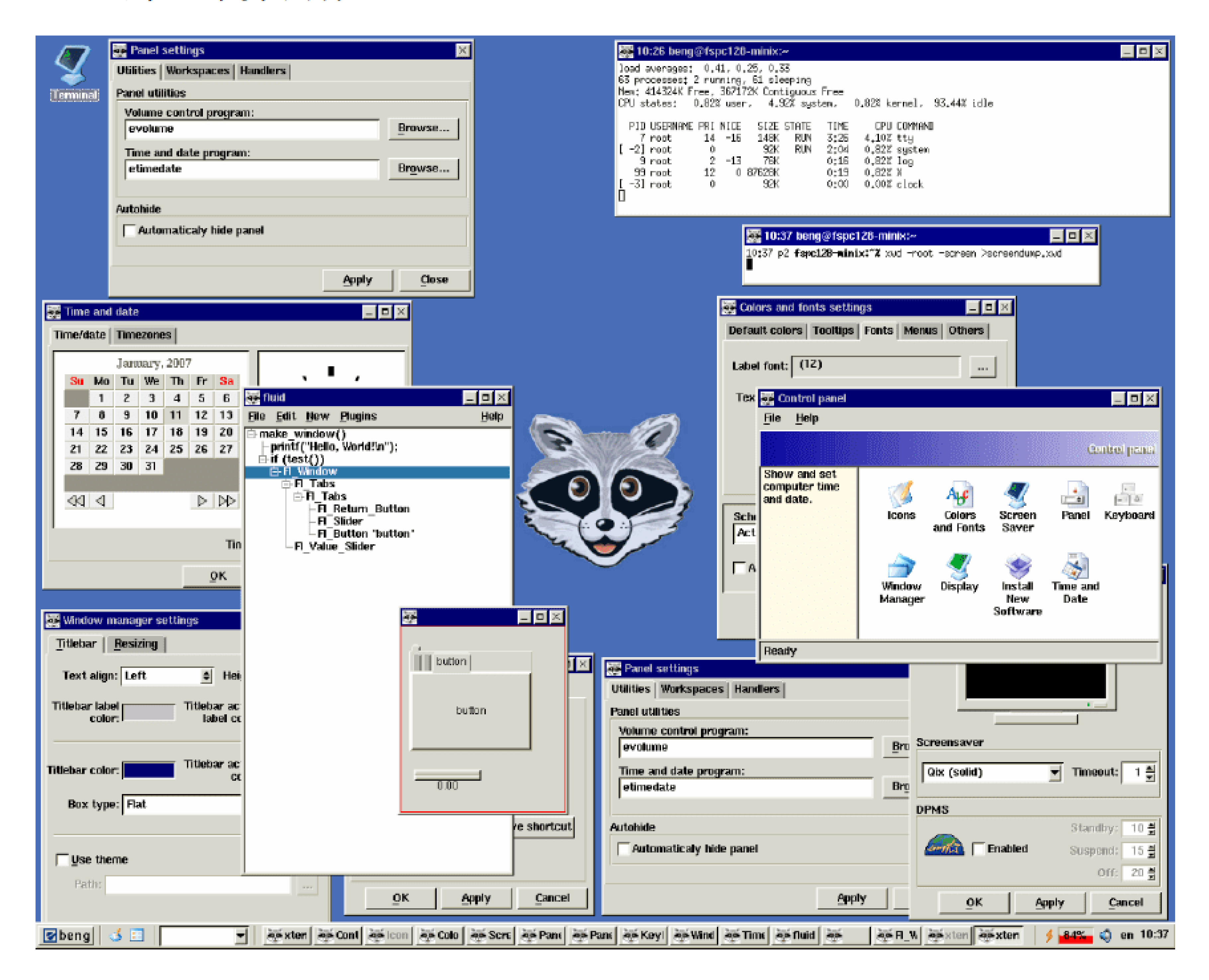


图 1.13　Minix 3 的界面

可想而知，很快这个教学用的操作系统就已经不能满足 Linus 大侠的欲望了，可是似乎也没有更好的选择。上面说过了，UNIX 奇贵无比，DOS 又不够优秀，而且无论 UNIX 还是 DOS，它们的代码都是不开放的，只能拿来用，没法拿来折腾。于是像其他牛人一样，Linus 自己动手了（当想要的东西不存在时就自己动手创造，这充分说明他有成为大牛的潜质）。

今天我们都知道，Linus 从那时起开始了一个事业，一个神话，但在当时，他并没有想那么多，只是为了学习 Intel 386 体系结构下的编程技术。他并不知道自己即将创造的是一个在世界范围广泛使用的系统，而只觉得是自己一时的异想天开。因此，一开始他把自己写的这个操作系统命名为 FREAX，有异想天开之意，就此开始了这个“异想天开”操作系统的编写。大约 1991 年 4 月份的时候，他就编写出了第一个可以运行的版本——0.00 版。这个版本可以启动，运行两个进程，分别在屏幕上打印出 AAA 和 BBB，然后……就没了。虽然连句整话都不会说，不过这是一个好的开始，至少能启动了。

提示：FREAX 源于英文中的 freak。freak 有怪诞，怪物之意，Linus 取其谐音命名其操作系统为 FREAX。

1.3.3　Linux 的由来

如果 Linus 就这么干下去，估计到今天只会有两种结果。

（1）成家立业后的 Linus 经常指着他的电脑 C 盘里面的一个文件夹对来访的朋友说：

看，我那时候还写过一个 FREAX 系统。

（2）Linus 为完成 FREAX 系统挑灯夜战，最终累得吐血身亡，永远活在我们心中。

总之，如果他一直自己干下去，就不会有 Linux 这个东西了，因为一个人的力量是有限的。有道是人多力量大，众人拾柴火焰高。Linus 深刻明白这一点，他没有独自在家闭门造车，而是让他的操作系统和互联网，亲密接触了。

“Hello everybody out there using minix——I’m doing a (free) operating system.”这是他当年在 comp.os.minix 上发布的消息，告诉大家，他正在写一个操作系统。并且，他还把他写的“异想天开”操作系统的代码上传到 ftp.funet.fi 的服务器上供大家下载，以便交流心得，共同学习。这就相当于你跑到网站上发帖子说：我研究出一种萝卜炖牛腩的方法，主料是啥啥啥，配料是啥啥啥，怎么怎么炖，大家都试试吧！（对不起，我又饿了）于是很多有兴趣的人就来尝 Linus 炖的牛腩，哦不对，是尝试 Linus 写的系统。不过当时那个服务器的管理员 Ari Lemke 看着这个异想天开的名字就不顺眼。想想，既然是 Linus 写的操作系统，又是类 UNIX 的，干脆，叫 Linux 吧。

这里先要说一个概念，Linux 是什么？狭义地讲，Linux 只是一个操作系统的内核，它只是各位的 Ubuntu 系统里面/boot/目录下的那个内核文件 vmlinuz-x.x.xx-xx-generic。就好比汽车，Linux 只是一个引擎而已，只是大家普遍习惯把装了 Linux 这种引擎的汽车叫做 Linux 汽车。那么既然 Linux 只是一个内核，要想工作，就还需要很多周边软件的支持，比如文件系统；比如一个命令行程序；比如一些基本的软件。这些东西加在一起成为一个系统，其实应该叫做 GNU/Linux 系统。不过为了符合平时习惯，本书后面如无特别声明，所提到的 Linux 都指 GNU/Linux 系统。

1.3.4　众人拾柴造就 Linux

Linux 被公布在网上之后，引来大家纷纷的路过和围观，很多人觉得这个东西挺有意思，不过第一个对外发布的 0.01 版 Linux 还有很多的不完善（这简直是一定的）。于是，全世界的有志之士纷纷伸出援手，共同完善这个刚刚出生的 Linux。

首先就要感谢 Richard Stallman 大牛创建的 GNU 计划，这使得 Linux 不必去从头开始开发那些最基本的软件和命令，而是直接利用 GNU 计划中的那些优秀的开源软件——前面说过了，那时候 GNU 系统除了内核以外，已经比较完善了。

有了基本的软件之后，还需要一个文件系统。由于当初 Linus 大侠是在 Minix 系统上开发的，所以最开始 Linux 用的文件系统是借用 Minix 的文件系统。可老借别人的总不是个事儿，还是应该有自己的文件系统，要不然你怎么好意思跟别的操作系统打招呼？这时候，来了个牛人叫 Theodore Ts'o，就是图 1.14 中这位。

Theodore Ts'o，曹予德，华裔，1990 年毕业于美国 MIT 大学计算机科学专业。他爱好广泛，喜欢烹饪、骑车、无线电报，还有折腾电脑（这些爱好都不挨着啊），当然这不是我们的重点。他看到 Linux 觉得很有意思，于是怀着极大的热情为 Linux 提供了邮件列表服务以便大家一起讨论问题，后来还提供了 ftp 站点来共享 Linux 的代码，并且一直用到现在。除此之外，技术上，他编写了 Linux 0.10 内核中的虚拟磁盘驱动程序和内存分配程序。在感觉到 Linux 缺少一个自己的文件系统后，他提出并实现了 ext2 文件系统，此后 ext

系列的文件系统一直都是 Linux 世界中事实上的标准，任何一个发行版都会默认支持 ext 文件系统，现在已经发展到了 ext4 了。

另一位牛人，一个英国人——Alan Cox，请见图 1.15，不要问我为什么牛人都不爱刮胡子，我也不知道。



图 1.14　Theodore Ts'o，曹予德

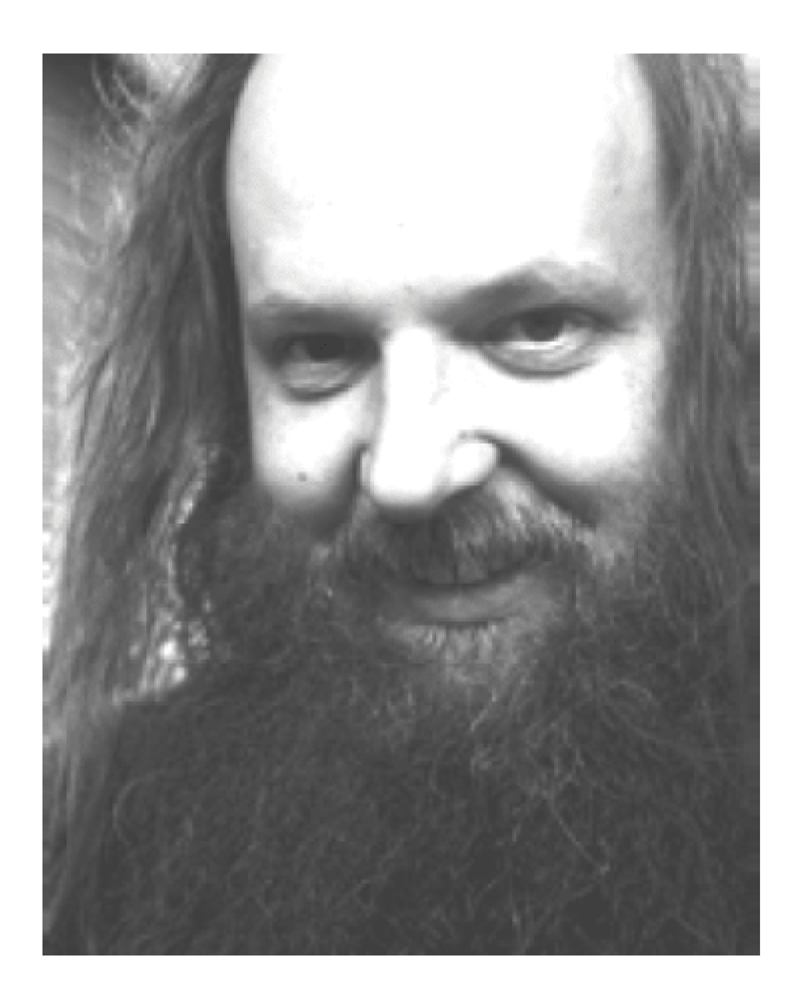


图 1.15　Alan Cox

他工作于英国威尔士斯旺西大学，特别爱玩电脑游戏（又一个玩游戏的，可见玩游戏也不是坏事），尤其是网游（你看你看，还是网游），不过那时候的网游不像现在这样华丽，那时候是字符界面的，能想象吗？字符界面的网游！那种网游叫做 MUD——Multi-User Dungeon or Dimension。玩 MUD 当然就得有计算机，得有网，所以 Alan Cox 开始逐渐地对计算机和网络产生了兴趣。为了提高电脑运行游戏的速度及网络传输的速度，他开始接触各种操作系统，为自己选择一个满意的游戏平台，争取榨干电脑的每一个指令周期。

经过仔细考虑，他买了一台配有 80386-SX 型 CPU 的电脑，并且装了 Linux 0.11 版的系统。这主要是因为预算比较紧张，即使是 Minix 他也买不起。于是他开始使用 Linux，进而学习其源代码，并对 Linux 产生了兴趣，尤其是网络方面相关的代码（整天琢磨怎么榨干他家那点带宽呢）。在 Linux 0.95 版之后，他开始为 Linux 系统编写补丁程序，以后逐渐加入 Linux 的开发队伍，并成为维护 Linux 内核源代码的主要人物之一。有一个稍微有点软的公司还曾经邀请他加盟，被他稍微有点硬地拒绝了。

提示： 80386-SX 是 Intel 公司于 1988 年年末推出的一款廉价版 CPU。其价格只相当于主流版本 80386-DX 的三分之一。

再有一位，Michael K. Johnson，他是著名的 Linux 文档计划的发起者之一，写了《内核骇客手册》一书，曾经在 Linux Journel 工作，现在就职于著名的商业发行版 Red Hat 的公司。

当然除了这些大牛，还有更多的大牛，中牛，小牛，肥牛……（唉，又饿了）他们都为 Linux 的发展作出了自己的贡献。他们来自不同的国家，从事不同的职业，甚至从未见过面。但是他们为了一个共同的目标，通过网络，一起合作，利用自己的业余时间，义务地帮助 Linux 成长，才有了今天这个可以合法免费使用的操作系统。这是什么精神？这就是“人人为我，我为人人”的软件精神！

1.3.5　琳琅满目的 Linux

这之后，Linux 的发展可以用“一发不可收拾”来形容。很多商业公司和民间组织都纷纷看好这个系统，并加入了 Linux 的阵营，各种各样的发行版满足着众多 Linux 爱好者的需求。

商业化比较成功的发行版，要数来自俄罗斯的 Red Hat 了，相信大家对图 1.16 里这顶红帽子有些印象。Red Hat 1.0 版于 1994 年 11 月 3 日发布，之后一直稳健发展。到 Red Hat 9.0 之后版本出现分支，其中的桌面版与来自民间的 Fedora 计划合并，成为 Fedora Core 发行版。而 Red Hat 公司则把精力全部投入企业使用的服务器版本——Red Hat Enterprise Linux。Red Hat 为 Linux 社区作的最大的贡献要数 rpm 软件包了，现在，相当多的发行版都使用 rpm 作为默认的软件包格式。



图 1.16　Red Hat 的徽标

Mandriva 是一个来自欧洲的发行版，它的前身是法国的 Mandrake Linux。Mandrake 的特点是方便，易用性好，硬件兼容性强。它为 Linux 的普及作出了很大贡献。2005 年 Mandrakesoft 公司与拉丁美洲最大的 Linux 厂商 Conectiva 达成了收购协议，Mandrake 从此更名为 Mandriva。Mandriva 和 Red Hat 一样，以 rpm 作为软件管理工具，部分兼容了 Red Hat Linux/Fedora Core 的软件包。图 1.17 所示是 Mandriva 的运行界面。

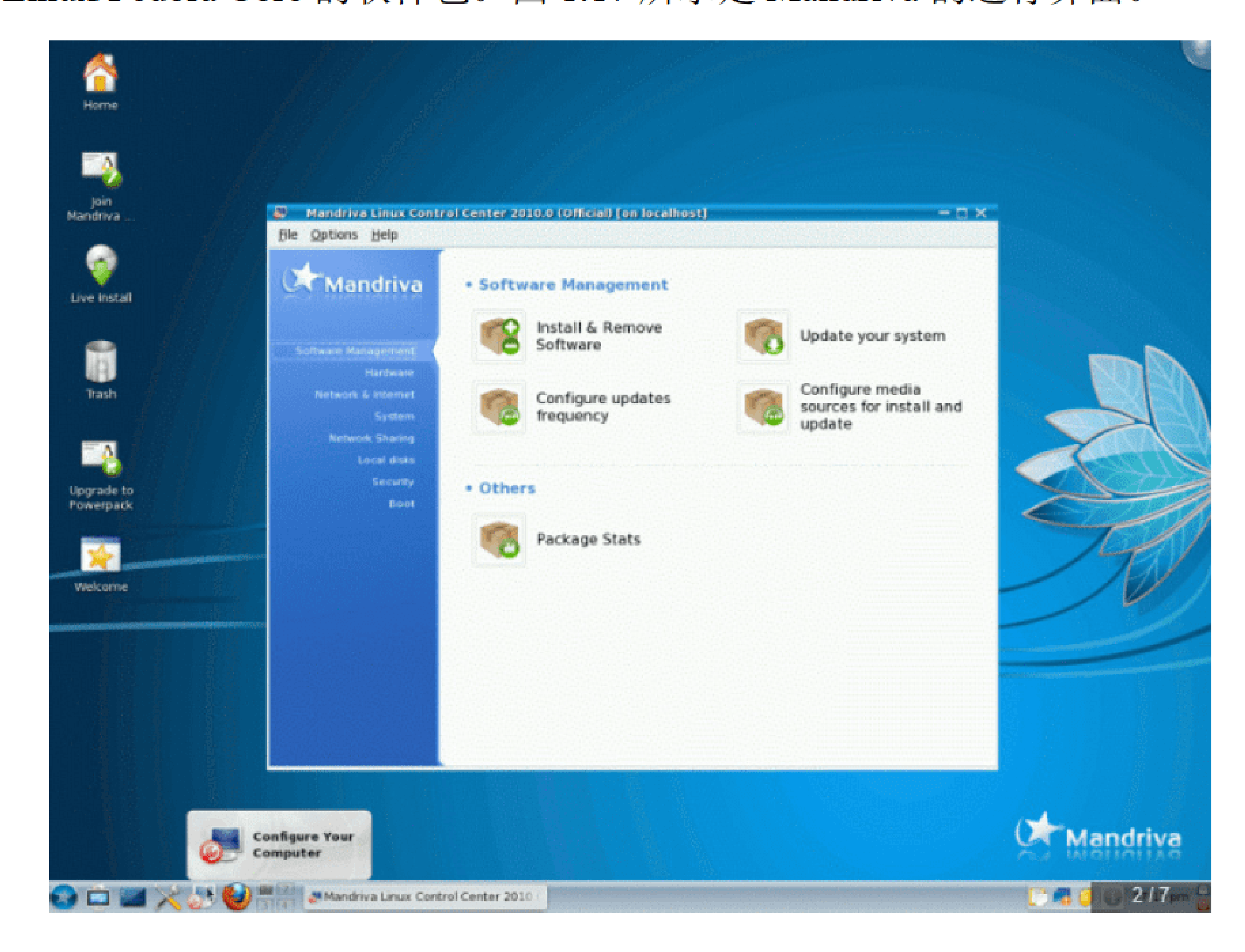


图 1.7　Mandriva 的运行界面

SUSE 是一个来自德国的发行版，隶属于 Novell 公司。它的特点是界面漂亮，但消耗的资源相对多一些，图 1.18 所示就是 SUSE 的界面。另外，它包含了一个安装及系统管理工具——YaST2。用户可以用这个工具进行软盘分区、系统安装、联机更新、网络及防火墙组态设置、用户管理等操作，为原来复杂的设置工作提供了方便的组合界面。

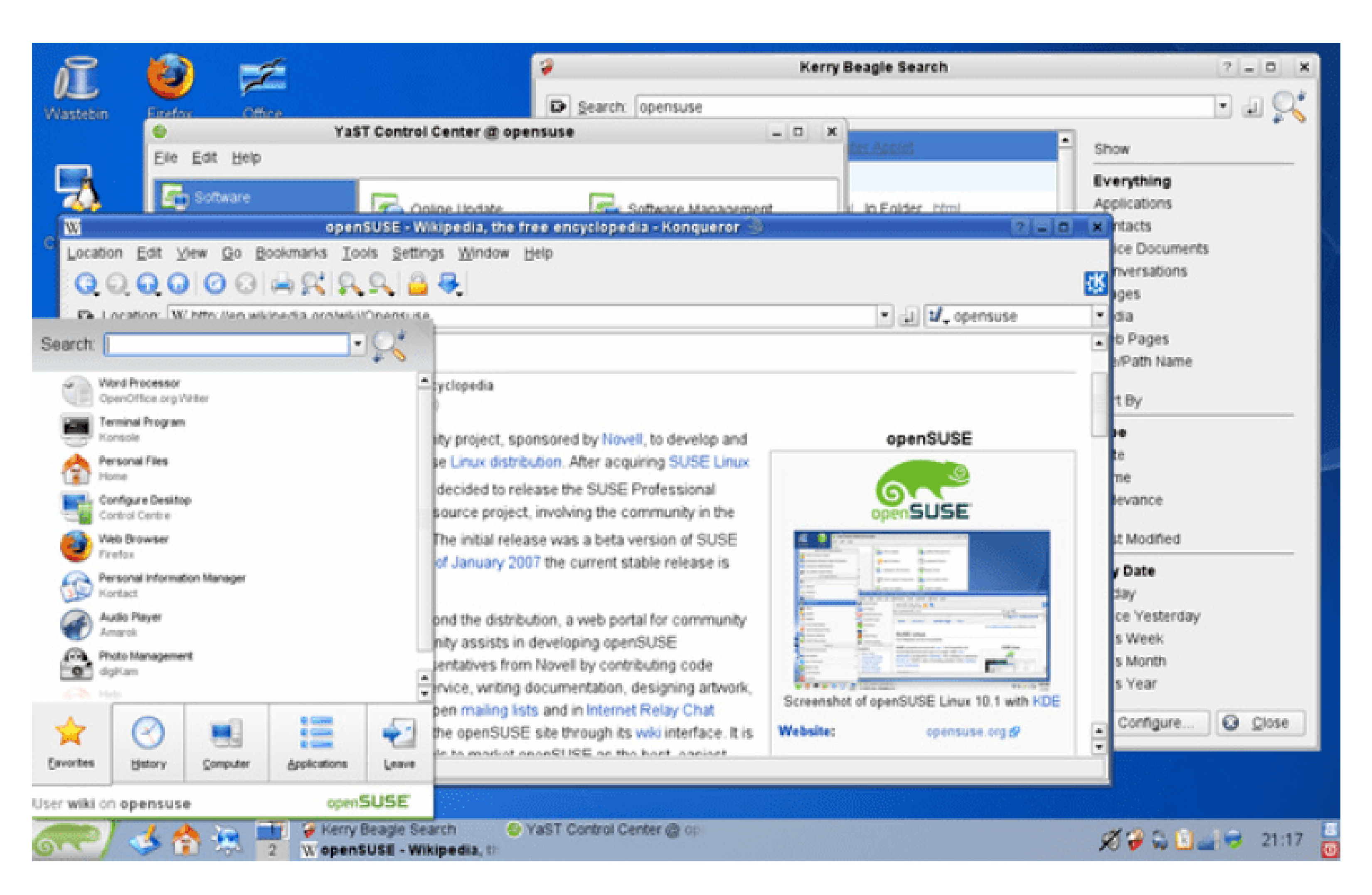


图 1.18　SUSE 界面

还有一个比较有历史的发行版，就是 Debian。它的徽标是一个豪放的螺旋，就像图 1.19 所示的这样。这是一个没有商业化，完全追随开源精神的发行版。Debian 于 1993 年发布第一个版本，一直到现在，路线没有大的变动，以稳定、保守著称。它的 deb 格式的软件包和 Red Hat 公司的 rpm 包具有同等重要的地位。同时，它的 apt 软件包管理器，也成为其他发行版竞相效仿的模范。



图 1.19　Debian 的徽标

除了这些之外，还有灵活的 Slackware、极端的 Gentoo、简洁的 Arch，以及我们这个故事的主角，在 Debian 的基础上改头换面而来的 Linux 界的新星——Ubuntu。

1.4　本 章 小 结

好了，Linux 家族的这点历史渊源就介绍到这里，相信您对 UNIX、Minix、Linux 这些发音差不多的系统，已经有了些简单的了解了。下一章，就该我们这本书的主角——Ubuntu 系统登场了。

第 2 章　初 来 乍 到

我叫 Ubuntu，我的用户喜欢叫我“笨兔”。但是我绝对不笨，与某种耳朵长尾巴短的哺乳动物也没有什么联系。我是一个操作系统，我是 Linux，我是 Ubuntu。

我是一只幸运的 Ubuntu，遇到了一个欣赏我，喜欢我，真正能够让我施展才能的用户。今天，我就讲讲我从出生，到被下载进电脑，直至安装到硬盘里的过程。

2.1　抵达——获得 Ubuntu 的途径

在 2010 年的 4 月，我来到了这个世界，并由出生的月份得到了我的代号——10.04，也就是 2010 年 4 月出生的意思。和我同一天出生的兄弟们还有很多，我们都是 Ubuntu 10.04。我们出生前一直在 Canonical 学校学习——这话您听了可能会费解，怎么出生前就开始学习了呢？

这不怪您，主要是因为咱们不是一个种族的。我们是软件，我们所谓的出生，也叫发布，也就是正式推出的意思。但我们在正式发布之前，其实就已经存在了，只是那时候还不完善，叫做 Beta 版或者 Alpha 版。我们出生前有很多的缺点：可能比较懒，经常启动不起来；可能脾气大，有点不顺心就吐一屏幕的英文字母给你看；要不就经不起打击，一不留神就死掉了（别害怕，我们软件死掉容易，复活也简单，重启就行了）。总之，我们出生前有很多的缺点和不足。所以要努力学习，争取到出生的那一天能够成为一个值得信赖的操作系统。或者不叫“出生”吧，听着别扭。这个过程其实有点像你们人类上学，那叫毕业吧？毕业，也就是正式发布的那天。

提示：Ubuntu 系统通常在每年 4 月和 10 月各发布一个新的系统版本。

2.1.1　毕业了，就要去工作

从 Canonical 学校毕业之后，我们就该参加工作啦。和你们人类一样，我们毕业的时候，也都充满理想；也都满怀激情；也都各奔东西；也都……不管分配。那我们怎么找工作呢？上招聘网？您见过招聘网站上提供下载操作系统的吗？我们是等着工作来找我们！

比如您想招我到您的电脑上工作，就可以从互联网大道走：到 www 市，Ubuntu 区，com 大楼，就找到我们学校了。还有中文分校，离着不远：也在 www 市，Ubuntu 区，org 大院里面的 cn 门。进去之后，您可以在里面随便逛逛，参观一下，了解 Ubuntu 系统的一些基本知识。之后按照页面上的提示，找到下载我们的网页，就可以开始下载了。一般中

文的下载页面是这里：

```
http://www.ubuntu.org.cn/download/ubuntu/download/
```

进到下载页面，会看到一个硕大的橙色按钮写着“开始下载”。不过先别着急点，先在左边选择好你需要的版本，就是图 2.1 中标示出的这个地方。

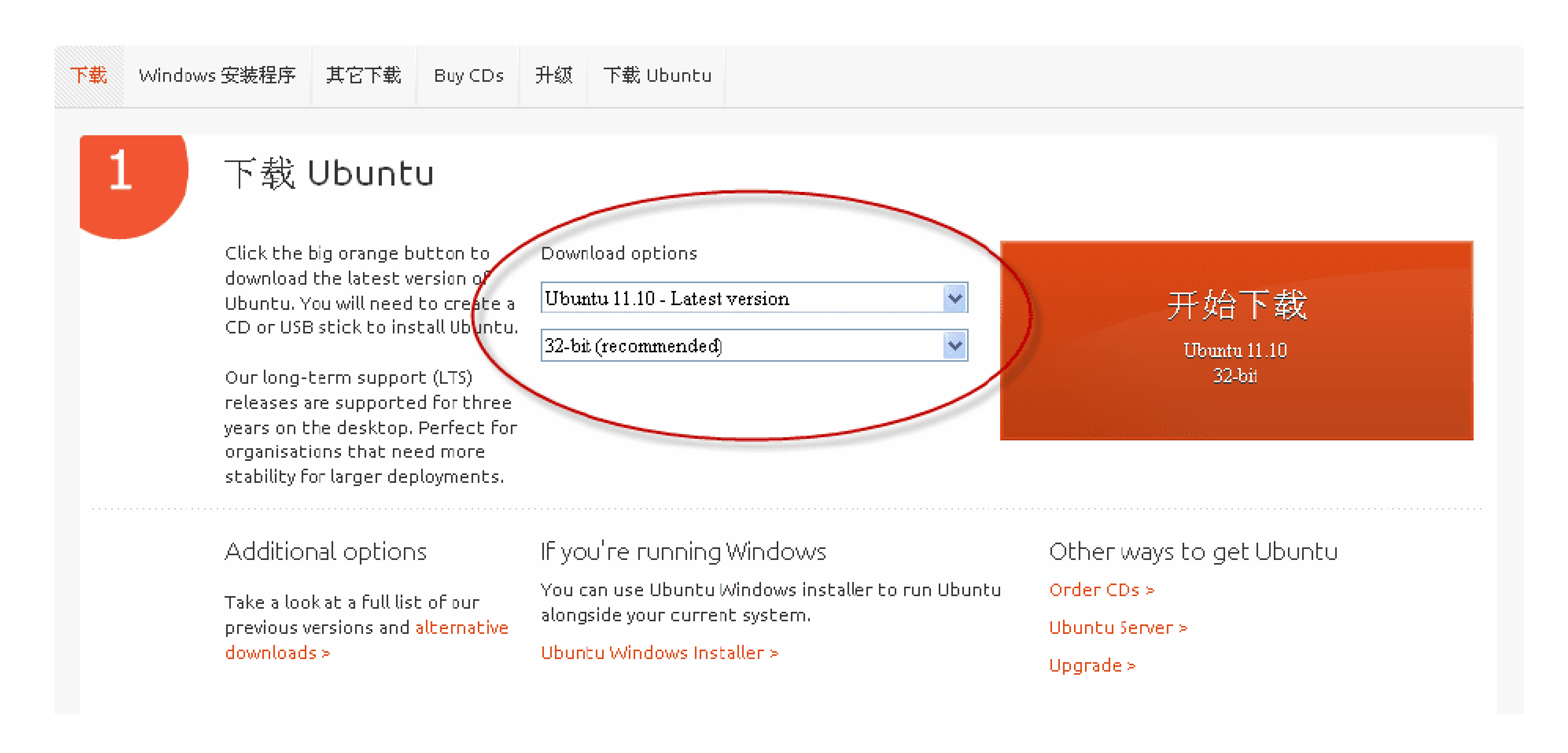


图 2.1　选择要下载的 Ubuntu 版本

图 2.1 中上面那个下拉列表框是选择我们 Ubuntu 系统的版本的。一般会有两个选项，一个是最新的普通版，另一个是最新的长期支持版（就是带 LTS 的版本）。普通版只提供 18 个月的支持，18 个月以后，虽然系统还能用，但是官方不再提供任何软件升级和补丁了。而长期支持版可以提供 3 年的支持，并且更加稳定，但是里面的软件就不是最新的了。

提示：无论普通版还是长期支持版，在有新的版本发布之后，都可以在线升级为最新的版本，如从 Ubuntu 11.04 升级为 Ubuntu 11.10。但只能相邻版本升级，不能跨版本升级。如从 Ubuntu 10.10 不能直接升级为 Ubuntu 11.10。

图 2.1 下面的下拉列表框是选择下载 32 位系统还是 64 位系统。虽然现在的电脑基本都是 64 位的了，不过还是建议一般用户选择 32 位系统。因为有些软件还没有 64 位的版本，所以使用 32 位系统遇到的问题会少一些。

选好之后单击那个硕大的“开始下载”按钮，就开始下载了。这里要说明一下，领我们回家是免费的，不需要花一分钱（当然，您自己的上网费自己掏），我们也不会因为您没花钱就隔三岔五地黑屏重启。

2.1.2　要工作，先要有住处

如果是从互联网大道把我拉回到您的电脑上的，那么您得到的是一个 ISO 文件，名字大约是 ubuntu-10.04.1-desktop-i386.iso、ubuntu-11.10-desktop-i386.iso 这样的。不过您如果只是把这个 ISO 文件静静地放在硬盘里，除了占用一点您宝贵的硬盘空间外是没有任何用处的。要想让我们为您工作，得把我们安装到您的电脑上，让我们在您的硬盘里定居才行。公司招俩研究生不还得给解决户口呢吗，我们也是这个道理。

提示：像 Ubuntu 10.04 这样长期支持的版本，每过一段时间会推出一个升级版，叫做 Ubuntu 10.04.1、Ubuntu 10.04.2 之类，就相当于 Windows XP 系统的 SP1、SP2。

要把我安装到您的电脑上有很多方法，最简单的就是把这个 ISO 文件刻录成光盘来安装。记住，要选择刻录镜像文件（刻录软件不同，叫法可能不一样，不过大概是这个意思）。可不要把 ISO 包解开，然后把里面一大堆乱七八糟的文件拖进去刻录；也不能直接把 ISO 文件拖进去，刻完了之后打开光盘一看，里面就一个 ISO 文件。这么刻完的光盘都是启动不了的。再次重复，要用镜像刻录，这样刻出来的光盘才能够正常地从光驱引导计算机，并进入安装界面。这也是自盘古开天地以来最通用、最正常、最安全的操作系统安装方法——光盘安装。

要想使用光盘安装，您的电脑先得设置成从光盘启动才行。这个大概不用我教您，一般打算安装 Linux 的人，设置 BIOS 启动顺序应该不成问题。设置好之后，把光盘放进去，重启就行了。就像现在，我就正静静地躺在一台电脑的光驱里，等待着和我的用户见面，等待着他启动电脑……

2.2　启动——安装 Linux 前的准备

在光驱中躺着，等着电脑启动的时候，心里总觉得有些忐忑，胡思乱想。估计刚毕业的大学生第一次面试之前也是这样吧。不知道我能不能被留在这个电脑里，我可不想被扔在角落里等着落灰。算了，不乱想了，抓紧时间整理一下第一次启动该做的事情吧，省得待会儿出错。

2.2.1　了解计算机的组成

作为一个操作系统，我应该对这个电脑的组成有所了解。好，那就来复习一下。

电脑，大名计算机，要说这可是个伟大的发明。它的出现极大地改变了人们的生活。最初的计算机个头很大，有一大堆这个管那个管，动不动就两个火车头，半拉四合院那么大。里面看上去很复杂，但功能相对简单。随着技术的发展，计算机的体积越来越小，速度越来越快。今天的计算机，看上去比以前简单（实际更复杂），但功能比以前强大了不知道多少倍。不过，虽然经过了复杂的演变，计算机的大体结构还是一样的。就像这年头盖的房子这么多，户型各式各样，但不外乎都有客厅、卧室、厨房、厕所。计算机也一样，不外乎都是由厨房、厕所……哦不对，不外乎都是由处理器、存储器和输入/输出设备组成的。注意，计算机里，没有厨房、厕所！

【处理器】

处理器，也就是我们常说的 CPU（Center Processing Unit，中央处理单元）。图 2.2 所示就是一块 CPU，大家都很熟悉。有道是“文臣纸笔安天下，武将刀马定乾坤”。无论干什么，总不能赤手空拳，多少都需要些工具。处理器就是我们软件工作的时候要用到的最重要的工具。每一个软件工作的时候都得用处理器，就好像会计工作得用算盘，厨师工作得用菜刀一样。处理器主要有计算和控制两大功能。

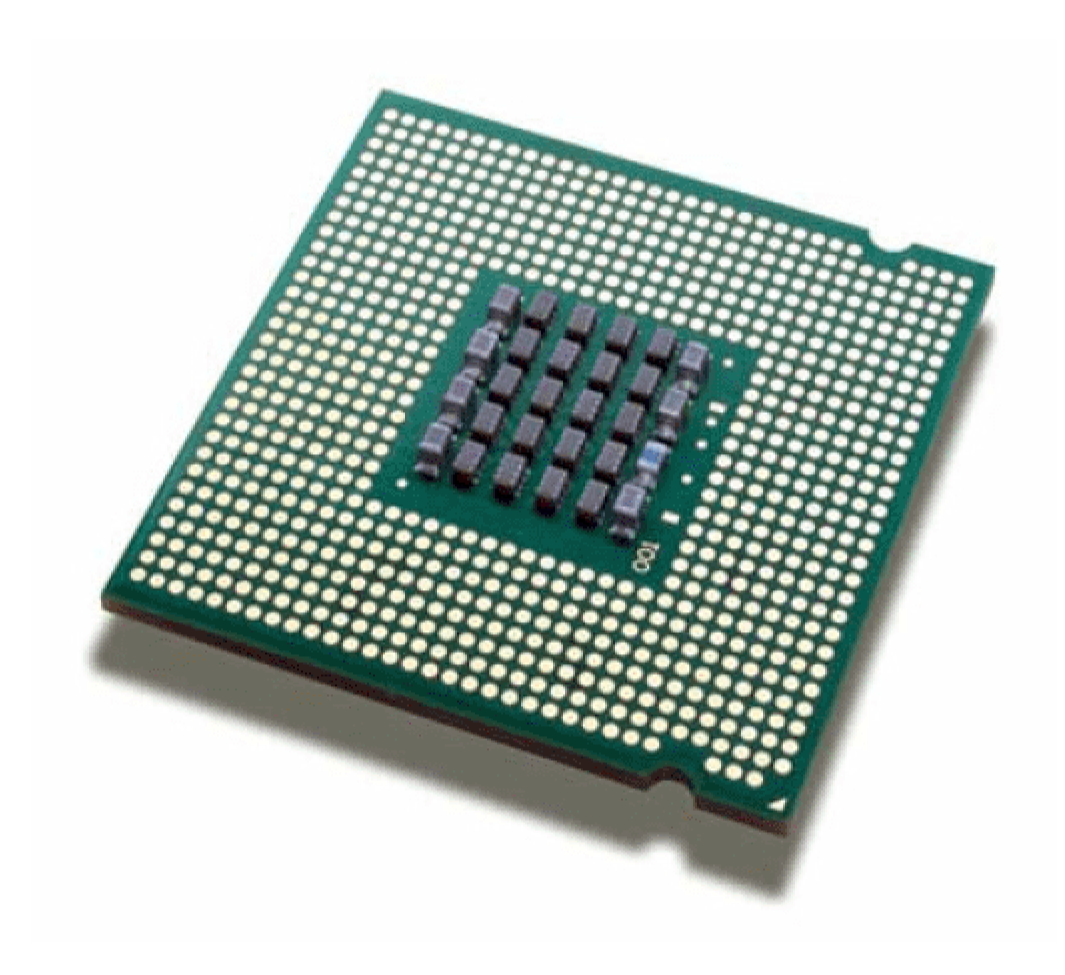


图 2.2　处理器

计算，好理解吧，就是算数。我们软件做任何工作都需要计算，也就是说我们软件做任何工作都需要用处理器。所以处理器的运算速度，直接影响着我们软件的工作效率。有句话怎么说来着？“工欲善其事，必先利其器”嘛。

那么处理器的控制功能又是什么呢？就是说计算机里的任何一个硬件，都直接或者间接地受处理器的控制。我们软件只要拿着处理器进行操作，就可以实现对声卡、显卡、网卡这些硬件的控制。这有点像汽车，虽然汽车要想行驶起来，需要各个零部件密切的配合，但是司机只要坐在上面，握好方向盘，踩对了油门、刹车、离合器，就可以控制整个汽车的行驶。

【存储器】

再说存储器，存储器就是用来存储程序的地方，换句话说，存储器就是用来给我们软件住宿、工作的空间。再说白点，软件待的地方，就是存储器！比如我现在所在的光盘，就是存储器。不过光盘只是我们软件从一台机器挪到另一台机器的时候所需要的交通工具，真正要定居在一台电脑里的时候，要住在硬盘里。有人说，那硬盘也是存储器了？没错，硬盘、光盘、U 盘这些都是存储器。

不过这些存储器有个共同点——都是程序们平时不工作的时候住的地方，它们都属于外存储器。而当一个程序真正要干活的时候是要到另一个空间去的，这个空间就是我们软件的工作间——内存储器，也就是大家常说的内存，比如图 2.3 所示这个，就是一个内存条。内存的大小对我们的工作效率也有很大的影响，内存越大，工作效率自然越高。你想啊，要是你们公司都坐得人挨人人挤人，恨不得把办公桌摞起来，老张坐老李脑袋上办公，那工作效率能高得了吗？

提示：最初的计算机内部没有磁盘，只有 RAM，因此磁盘——包括硬盘、软盘，都算做外存储器。

【输入/输出设备】

那么这个输入/输出设备是干什么用的呢？咱回过头来想想，有了外存了，我们软件的住宿问题解决了。有了内存了，我们有了工作间了。然后又有控制器了，我们有工作的工具了，好，可以开工了！等等，先别急，您想想咱开工干什么啊？得有人给我们任务呀，

要不我们拿着 CPU 算什么呢？不能自己算“1+1=2”玩吧。那么任务是谁给我们的呢？当然是坐在电脑前的人了。

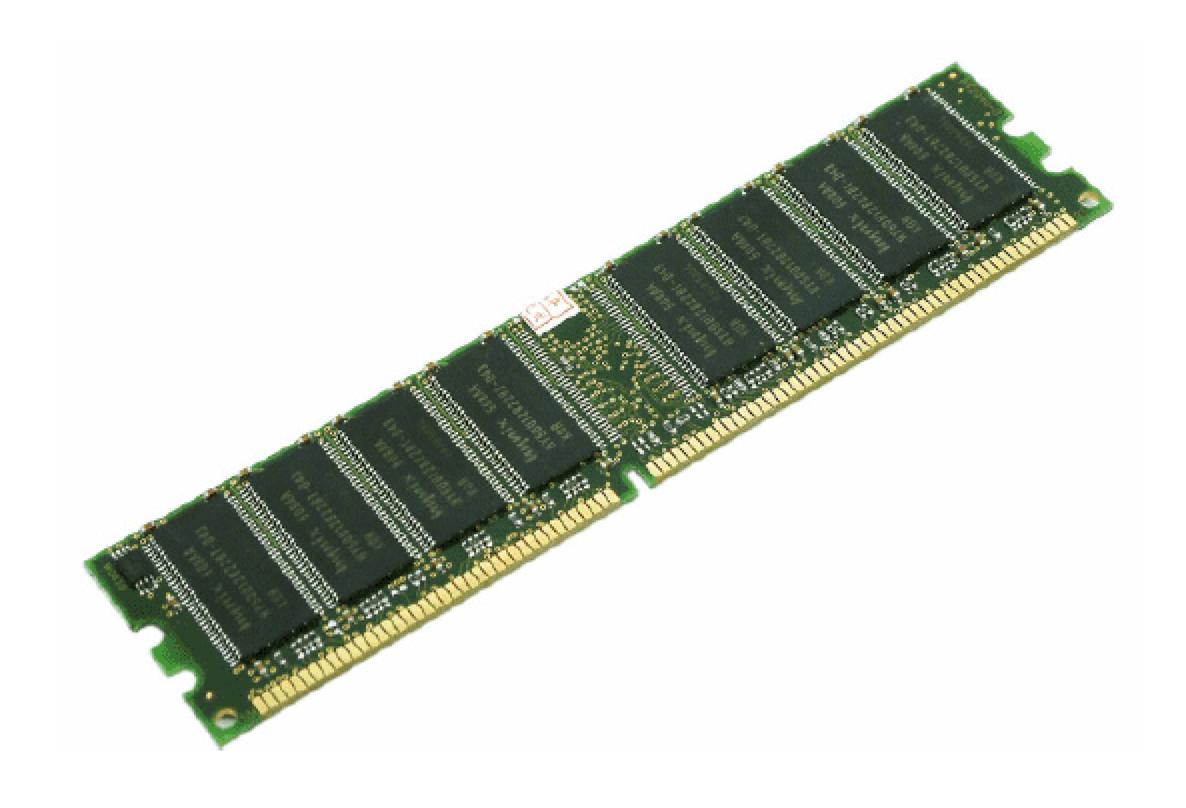


图 2.3　内存储器

可是，有句话叫人鬼殊途。软件虽然不是鬼，但和人类也是不能直接对话的。所以，我们之间的交流需要设备。输入设备就是用来让人类给我们发指令、分配任务的。比如鼠标、键盘、触摸屏这些都是输入设备。输出设备相反，是用来让我们软件计算得出结果后把结果反映给人类的。显示器、音箱、打印机是输出设备。

我正在整理着思路，忽然感到一阵震动。之后，光盘缓缓旋转，逐渐加速——终于启动了！

2.2.2　先尝后买——用 LiveCD 体验 Ubuntu

这时候电脑正在从光驱引导，光驱里那扇通往内存的 IDE 通道之门已经打开，我背起我的背包——一个 RAM 文件系统，走过 IDE 通道，进入这台机器的内存里。

【第一印象】

进入内存里之后，我把 RAM 文件系统展开，把里面的东西掏出来放好。背包是我来的时候就打好的，里面是一个能够运行起来的文件系统，包括很多跟我一起干活的同志们，如 Firefox、apt、gnome 等。把文件系统搞定之后就叫这帮人起来干活。哦，对，还有驱动，赶快翻翻我带来的驱动程序。我们学校的老师教导我们说，第一印象是很重要的，所以一定要在 CD 上面的 RAM 系统包里打进尽可能多的驱动程序，这样无论遇到什么硬件，都能够直接地正常使用。否则用户从光盘一启动，就发现分辨率混乱，音箱不出声，那就麻烦了。

马上要显示出界面了。这是我第一次与用户面对面交流，不免有些紧张。听我的学长们说，一般我们 Ubuntu 系统在第一次运行后会有两种结果：可能我的能力会被认可，我会被安装在这台计算机中，实现自己的价值；或者，在一次不愉快的试用后，连同我乘坐的光盘一起，被扔到一个不知名的角落，或者给用户家的宠物狗当飞盘玩。好吧，不管未来怎样，我现在都要尽自己最大的努力，展现出我最好的一面。

终于，我收拾好了所有的东西，从光盘里来到了内存中。赶紧向还在光盘里的弟兄们汇报一下：“我已出仓，感觉良好……”嘿嘿。哦，对了，用户还在那等着呢，赶快显示出启动界面，如图 2.4 所示。

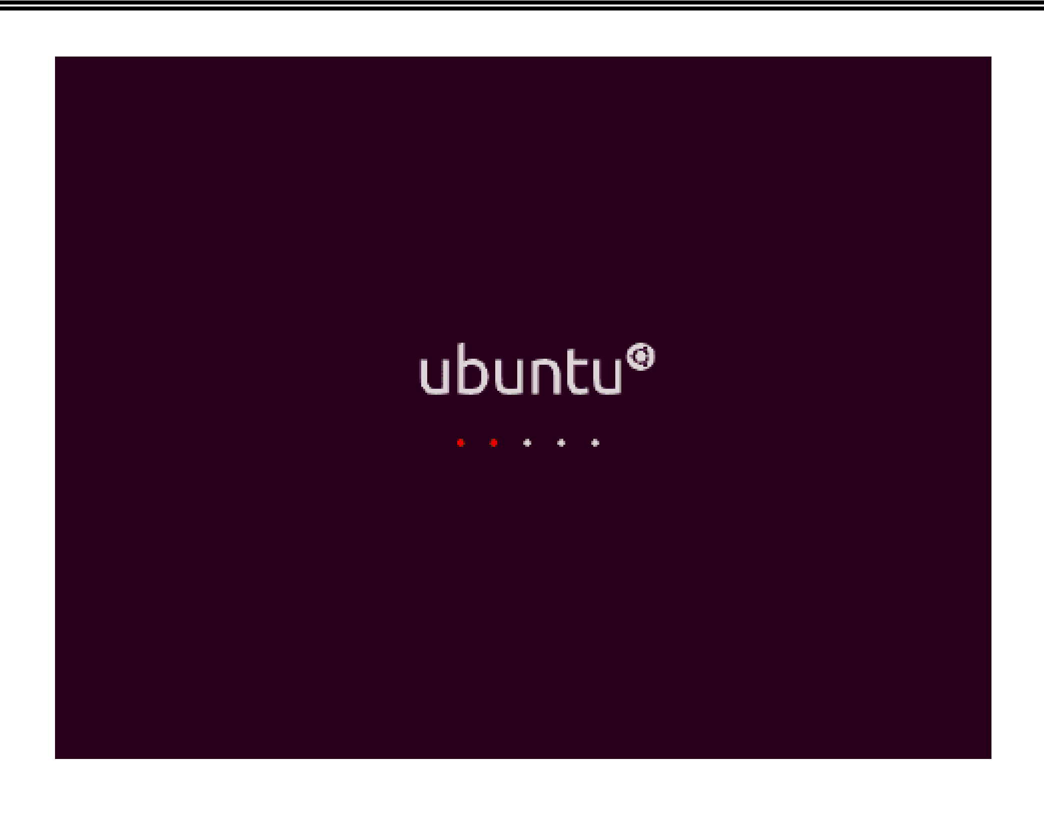


图 2.4　Ubuntu 的启动界面

【先体验，后安装的 LiveCD】

启动界面过去之后，首先要确定一下交流的方式。这很重要，就像你走在大马路上看见一个高鼻梁黄头发蓝眼睛的家伙，不可能过去就拍人家肩膀问：“吃了吗您？”智力正常的人一定是先过去来句：“Excuse me？”（除非你知道这人就是你家隔壁那隆过鼻子，酷爱染发，老戴对美瞳彩片的二嘎子）。但我们软件是无法看到使用者眼睛颜色的，所以我只好像图 2.5 这样，在屏幕左侧的列表框里列出所有我可以使用的语言，让用户来选择。



图 2.5　语言选择

这位用户毫不犹豫地在列表框里选中了“中文（简体）”。看来这家伙是个中国人，于是我马上转换到中文跟用户交流。首先我问他想要干什么，我给出如下两个选项。

（1）“试用 Ubuntu 10.04.1 LTS”——这个选项的意思就是先尝后买，好不好用得先试试，看着顺眼了再装。新手一般都选这个，能先看见我这系统到底什么样，心里有底了再装。

（2）“安装 Ubuntu 10.04.1 LTS”——这不用我说了，意思就是安装，这个选项一般是心里有底的老熟人选的。

可能有人对第 1 个选项还是不理解。试用？这系统还没装呢就能试用？对，能！因为我们 Ubuntu 的安装光盘是一张 LiveCD。那么什么叫 LiveCD 呢？

所谓 LiveCD，就是直接从光盘就能启动电脑并且运行的系统。整个系统在光盘上，启动后从光盘读取到内存里工作，可以进行一些基本的操作，像上网、听歌、玩游戏什么的，完全不需要硬盘。

通过 LiveCD，就可以在安装之前，先对系统有个体验，也可以测试一下计算机的硬件是否都能很好地支持这个系统。如果哪天系统出问题了，还可以用这张光盘启动计算机，对硬盘上的系统进行修复（类似于 WinPE 的功能）。用户试用之后如果觉得好用，想安装了，就可以双击桌面上的 Install 图标，把系统安装在硬盘上了（这跟刚才直接选安装是一样的）。

好了，这个使用者像大多数人一样单击了“试用 Ubuntu 10.04.1 LTS”按钮，于是我去叫醒和我挤在同一张光盘里的兄弟们：哥儿几个，考验我们的时候到了！

提示：有一些专门专注于 LiveCD 的 Linux 发行版，比较有名的有 Puppy Linux、Knoppix、Slax 等。

【不算高的配置需求】

按照指示，我开始进行系统启动的准备工作。

首先要检查一下这个电脑的硬件配置，如果 LiveCD 可以启动到桌面，并且速度不是太慢，说明这台电脑的硬件配置基本符合安装的要求。

“什么？你们 Linux 不就是跟 DOS 似的系统吗？也对硬件有要求？”对此，我只能说，您好像 OUT 了。虽然我们 Ubuntu 系统对硬件配置的要求一般，不算高，可也不能太低了。尤其我们 10.04，怎么也得用 2000 年以后的机器吧。液不液晶无所谓，主要得看机箱里边。像 CPU，怎么也得 1 GHz 以上吧，你弄一个 800 MHz 的奔腾 3 代，也好意思跟我打招呼？！内存 512 MB 起，硬盘怎么也得 5 GB，什么办公的、作图的、聊天的，能装的软件我全都得给你装上呢。还得有个网卡，无线的有线的都行，ADSL 拨号还是接路由的随便，反正得有网。要是没有网络想装 Ubuntu，装好了也急死你。

提示：可以到这个网址查看 Ubuntu 系统所需硬件基本配置要求：https://help.ubuntu.com/10.04/installation-guide/i386/minimum-hardware-reqts.html。

【还算广泛的硬件支持】

除了检查电脑的配置，还要扫描一下这里的所有硬件，以确定加载哪种驱动程序，把能驱动的硬件都驱动起来。因为 LiveCD 试用的过程也是检查我们 Ubuntu 系统的硬件兼容性的过程。用光盘启动电脑一看，硬件都正常工作，这就放心了，说明这台电脑装 Ubuntu 没什么问题，直接装上都不用装驱动。否则可能就要在安装系统之后再上网找驱动安装了。

话说我所在的这台机器条件还不错。4GB 的内存很宽敞，硬盘也有 500GB 大，其他的主要硬件，我也很熟悉。这主要是因为我们在 Canonical 学校的时候就进行了充分的学习，所以这里的东西我基本上都会用得比较顺手。像大螃蟹公司（特指 Realtek 公司）的网卡啦、Intel 公司的南桥、北桥、声卡及双核 CPU，我都能应用自如。只是这里的显卡是 Nvidia 公司的一款独立显卡，目前我还不能完全驱动它。不过别急，我们这一届 Ubuntu 系统，专门着重学习了显卡的使用，虽然默认情况下还是不能够启动 3D 加速，但是 2D 的显示已经很顺畅了，不会影响使用。要想启用 3D 加速也不难，等系统装好之后再去安装驱动就可以了。

提示：想查看自己的硬件是否兼容 Ubuntu，可以访问这个地址：https://wiki.Ubuntu.com/HardwareSupport。

检查得差不多了，我从光盘上叫醒了图形部门的哥儿几个，主要是 X Window 和 Gnome 小组，这两个部门负责在屏幕上显示图形操作界面的任务，以后还会经常介绍到他们。兄弟们干活儿都很麻利，只是光驱转得有点慢，所以耽误了一小会儿之后，屏幕上终于显示出了我们 Ubuntu 的默认桌面，如图 2.6 所示，应该不算土气了吧。

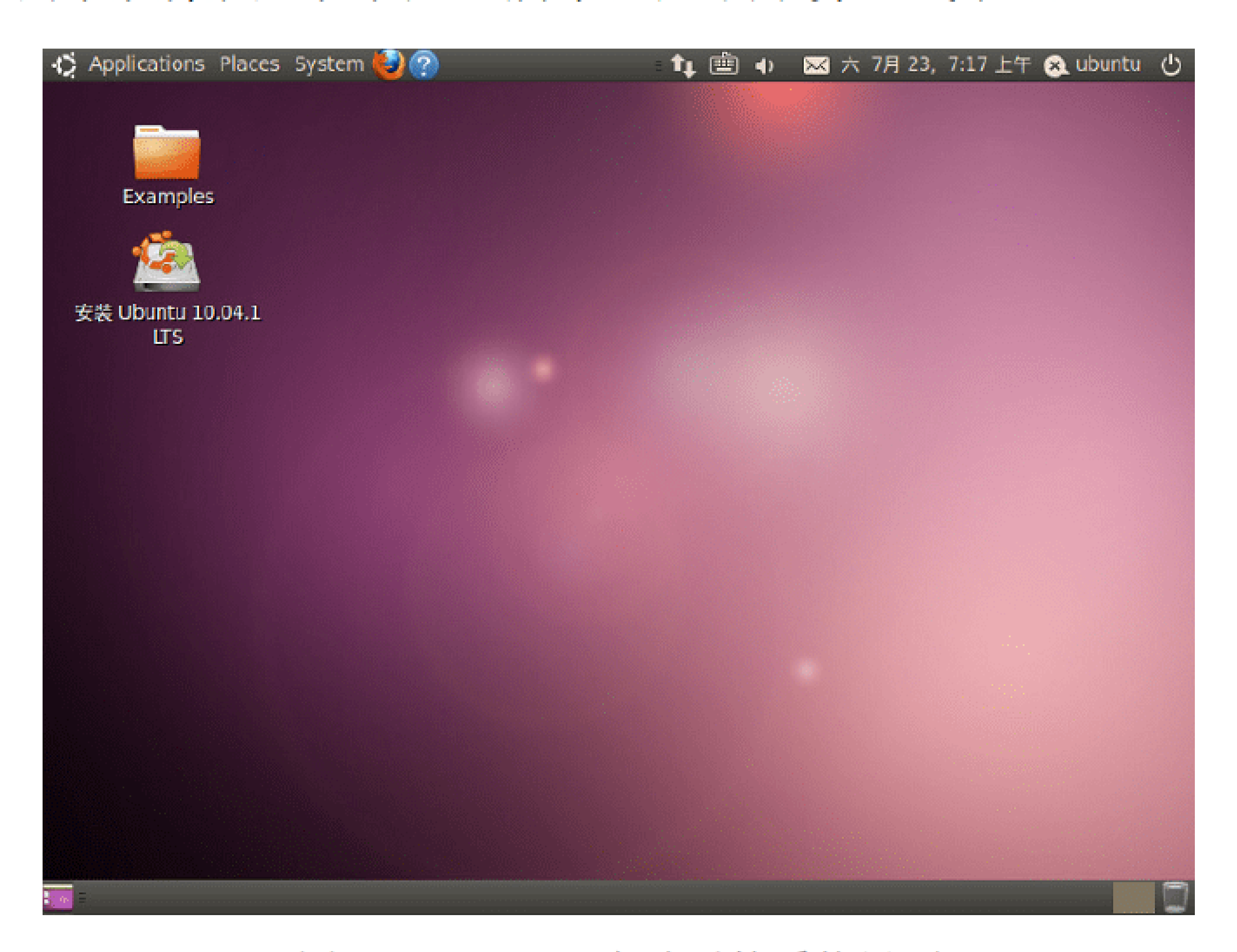


图 2.6　LiveCD 启动后的系统界面

启动了之后，用户很好奇地点来点去。先是玩了会儿游戏；又看了看光盘里的一些示例文档；再打开 Firefox 上了会儿网。最后终于下定决心，双击了桌面上那个“安装 Ubuntu 10.04 LTS”的图标——装！

2.3　入　　住

终于开始安装了，我要住进这台电脑啦！安装一共有 7 步。

2.3.1　第 1 步：选择语言

跟启动一样，还得先问一下用户打算使用什么语言，如图 2.7 所示。有人说了，你这家伙健忘吧，刚才不是选过了吗？别急，听我解释。刚才选择的是从光盘启动的 LiveCD 系统使用什么语言，这回选择的是安装到硬盘的系统用什么语言。当然，我也不傻，知道一般情况下这两个都是一样的，所以给用户默认选择了简体中文。然后，用户只要单击“前进”按钮，这一步就算完成了。

2.3.2　第 2 步：选择时区

选择时区也简单，根据主人选择的语言，我估计是个中国人，虽然这个国家地方大，

不过全国都是一个时区，因此我替他选择了亚洲区，中国，上海——也就是东 8 区，如图 2.8 所示。好了，他继续单击“前进”按钮。

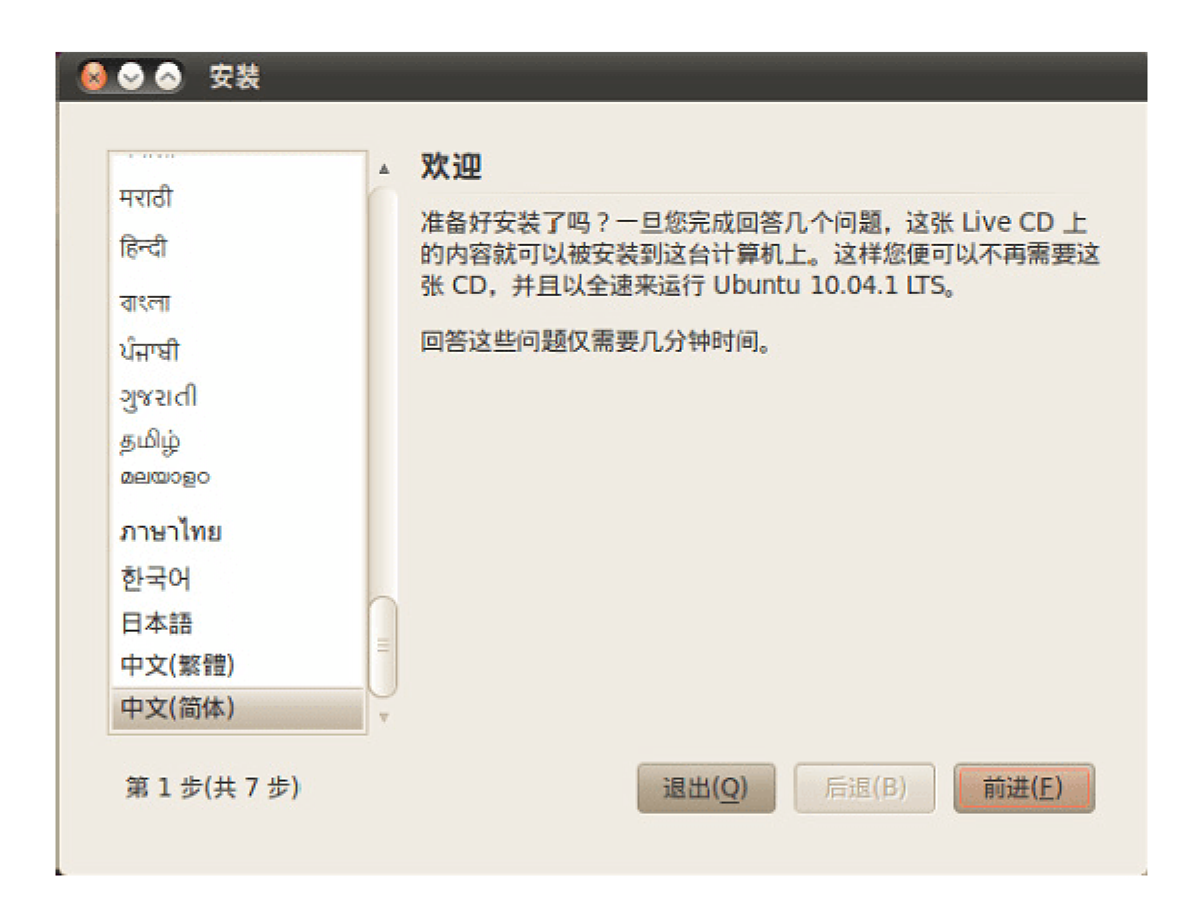


图 2.7　选择安装系统的语言



图 2.8　选择时区

2.3.3　第 3 步：选择键盘布局

这个我也代劳了，替他选择了美式键盘，多数情况下是没错的。如果不是，可以单击“猜测键盘布局”单选按钮，然后单击后面的“猜测…”按钮，之后按照我的提示按下一些按键，我就可以知道用户用的是什么键盘了。或者，用户也可以自己直接在下面的列表框里选择自己的键盘。如果不知道自己选得对不对，我还给他提供了一个文本框来测试，就是图 2.9 所示的那个，多体贴啊！好，我的这位用户还是直接单击了“前进”按钮。



图 2.9　选择键盘布局

2.3.4　第 4 步：分区

这回要了亲命了……好，咱一点一点慢慢说。到第 4 步分区这里，首先会有几个选项，让你选择分区的方式。

【清空并使用整个硬盘】

我最喜欢的是“清空并使用整个硬盘”，就像图 2.10 这样。这个选项的意思就是说不管现在硬盘里住着谁，有什么东西，统统给我卷铺盖走人，爷要住了！（低调低调）这样装完了之后硬盘上就啥都没有了，只剩我一个 Ubuntu 系统。不过一般人不会选这个选项，因为多数情况下硬盘里已经住了一个系统，而且用户并不想赶他走。

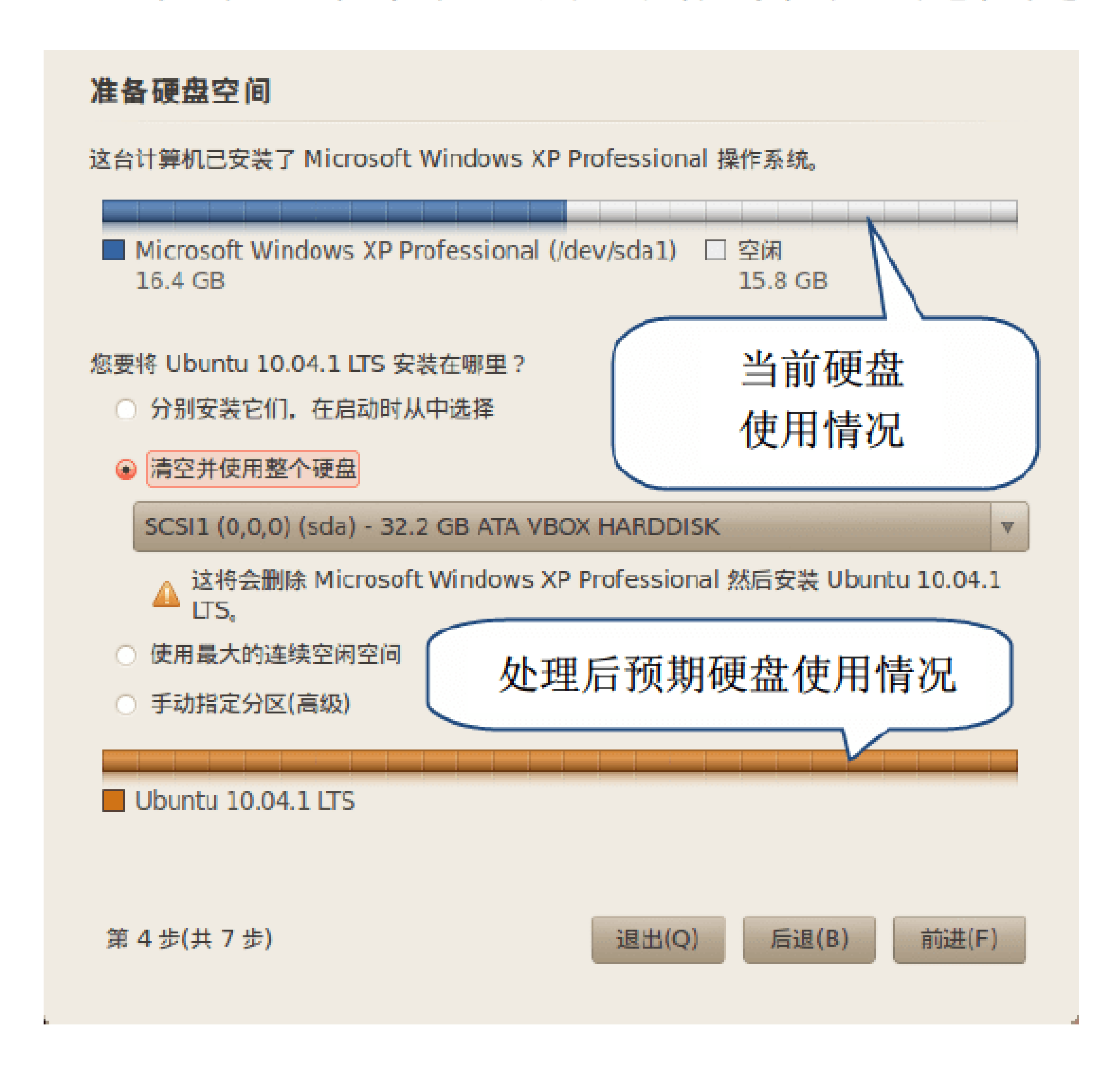


图 2.10　清空并使用整个硬盘

【分别安装它们，在启动时从中选择】

还可以选这个选项：“分别安装它们，在启动时从中选择”，如图 2.11 所示。选了这个选项你就什么也不用管了，一切交给我来处理，我办事，你放心。用户可以做的就是调整一下给我这个系统分配的空间，然后我自己会修改你已有的分区大小，挤出足够我住的地方来并安装。

提示：这种改变分区大小的动作还是有一定危险性的，请谨慎使用。



图 2.11　分别安装它们，在启动时从中选择

【使用最大的连续空闲空间】

比较没危险的就是这个“使用最大的连续空闲空间”。意思就是说，已经在硬盘里住下的系统不去管他，空间也不用调整。有多少地方空着呢，我就去那里整理整理住下。不过要注意，这个“空着”，可不是指你的 E 盘或者 D 盘之类的有空闲空间就行，而是得有没分区的空间，就像图 2.12 中的“未指派”的那样的区域才行。所以，要选这个选项，你需要事先在硬盘上空出一部分空间来不分区，或者把已有的分区删掉一个才行。

以上几个选项都是自动分区的，也就是由我自己决定划分多大的分区出来，哪个分区作为“/”等。最后一个选项就是“手动指定分区”，这个选项就需要了解分区知识的用户才能用，所以后面写了个“（高级）”。好，现在咱就仔细说说手动分区。

【手动分区】

选择了“手动分区”单选按钮并单击“前进”按钮之后，首先会显示出电脑当前的分区状态，如图 2.13 所示。同时，还可以注意到，总共的步骤变成了 8 步。多出了一步手动分区的步骤，也就是你现在正在操作的步骤，后面的步骤依次顺延。好，来看看怎么分。

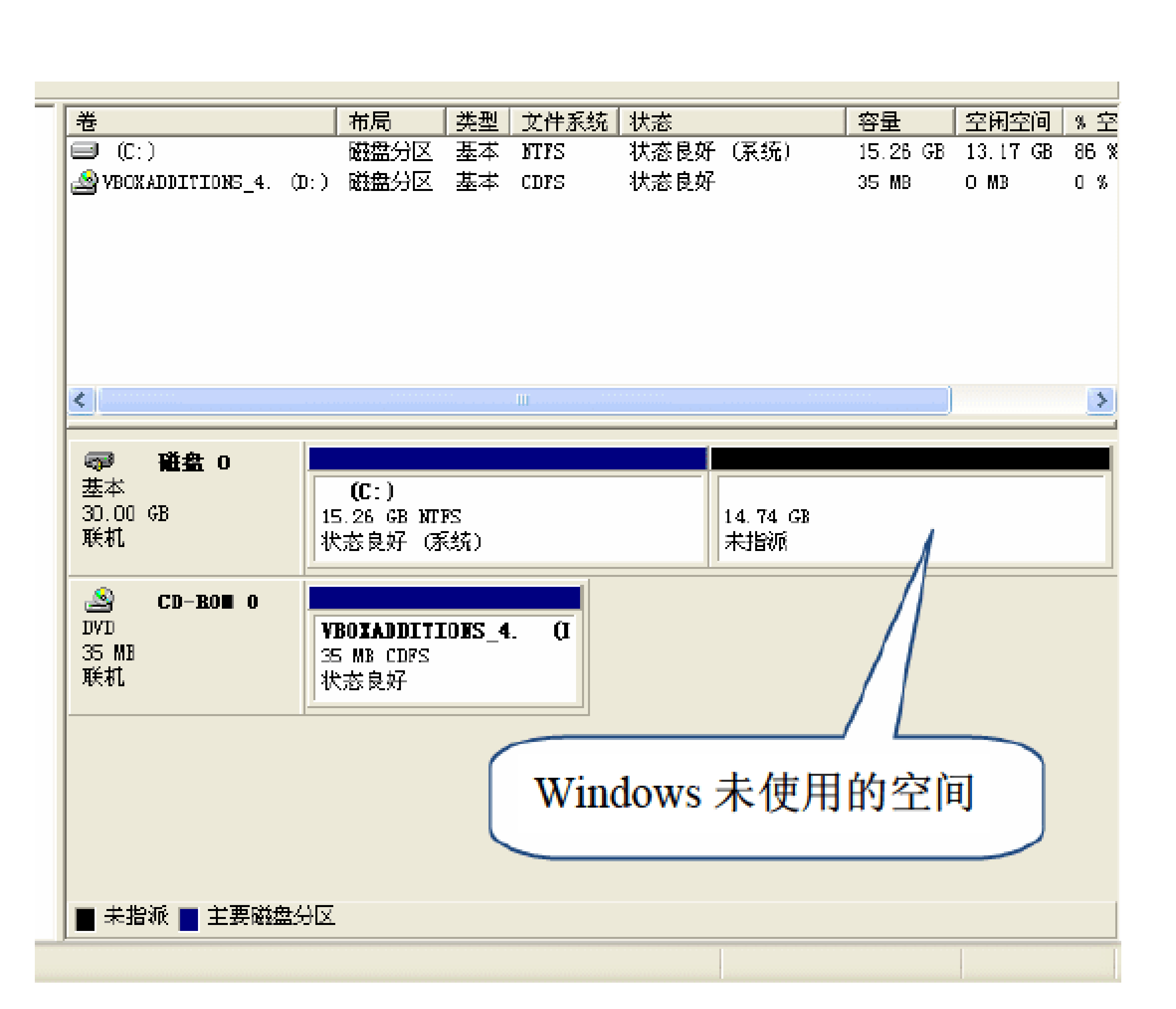


图 2.12　Ubuntu 需要的硬盘空间



图 2.13　手动分区

上方用一根棍状物体表示的就是你的硬盘，里面可能已经有一些不同种类的分区，用不同的颜色表示，下面用文字具体描述了当前分区的状况。有人会问，这个/dev/sda 是什么啊，看着怎么这么奇怪呢？好，凑近点看，如图 2.14 所示。



图 2.14　硬盘和分区的表示方法

不必奇怪，/dev/sda 是我们 Linux 用来表示硬件的方式。/dev/是一个目录，你看这个名

字，dev，就是 device 的简写，这个目录下面放的全是设备文件（在我们 Linux 世界里，什么东西都是文件，硬件设备也是文件）。这个/dev/sda，就是/dev 目录下的 sda 文件，这个文件代表什么呢？代表你的硬盘！sd 代表存储设备，可能是硬盘，可能是 U 盘，也可能是 SD 卡之类的，a 代表第 1 块，那么第 2 块就是 sdb，第 3 块就是 sdc。如果不考虑 U 盘之类的移动存储设备（假设安装的时候没插着），那么/dev/sda 的意思就是你的电脑上的第 1 块硬盘。要是第 2 块，那自然就是/dev/sdb 了。那么下面那个/dev/sda1 呢？就是/dev/目录下的 sda1 文件，这个文件代表你的第 1 块硬盘上的第 1 个分区。那么第 1 块硬盘上的第 2 个分区就是/dev/sda2，第 2 块硬盘上的第 4 个分区就是/dev/sdb4。

提示： 旧版内核还会区分 IDE 硬盘和 SATA 硬盘，IDE 硬盘的设备文件为/dev/hdx，而 SATA 硬盘的设备文件为/dev/sdx，现在都统一为/dev/sdx。

那么现在，选中列表框里面“空闲”的硬盘空间，然后单击“添加”按钮，来添加一个分区。如果没有空闲呢？那就找一个你看着不顺眼的分区删呗！（上面的数据丢了别赖我啊，记得提前保存好）。添加分区的步骤并不复杂，就下面这么几步。

（1）单击“添加”按钮之后，会出现一个如图 2.15 所示的“创建分区”对话框，在其中可以选择分区类型是主分区还是逻辑分区。如果你不知道这两个选项有啥区别，那就不要动，用默认的就好，反正我们 Ubuntu 系统装在什么分区都可以。

（2）其次是分区容量，这个不用多解释吧，在文本框里写上希望的分区大小就可以了，单位是 MB。

（3）分区的位置，一般也不用改，用“起始”就好。

（4）选择文件系统，也就是图 2.15 中的“用于”那个下拉列表框。我们 Linux 是不能使用 FAT 或者 NTFS 这样的文件系统的，这都是 Windows 系统用的。我们可以用的文件系统很多，新手可能不知道选哪个好。关于每个文件系统的区别和特点，来日方长，有机会慢慢说，现在，如果你不知道选啥，那就还是选默认的 Ext4 日志文件系统好了。

提示： 安装时可选的其他几种文件系统中，reiserfs 对于存取大量小文件的操作效率较高；XFS 对大分区、大文件的操作有优势；btrfs 对 SSD 硬盘做了一定的优化。

（5）挂载点，是让你指定这个分区用来干什么的。必须有挂载点是“/”。也就是说，你至少要分出一个区，挂载点选“/”。这个区有 5 GB 就够安装系统的，不过如果你想拿我们 Ubuntu 作为日常使用的系统，而不是只装来玩玩的话，大方一点，分 20 GB 吧。

好了，全都选完了，单击“确定”按钮，一个分区就建好了。

创建了“/”分区之后，再创建一个交换空间，步骤同上，只是在选“用于”的时候选“交换空间”，就行了，挂载点不用选，如图 2.16 所示。交换空间就相当于 Windows 下的虚拟内存，它的大小大约等于内存的大小就可以。如果内存很小（1 GB 以下），交换空间最好是内存的两倍。

提示： 对于 2 GB 以上的内存，日常应用基本不需要交换空间，只会在休眠的时候用到。因此即使不分交换空间，系统也是可以工作的。

好了，分了“/”分区和交换空间就可以继续进行了。但是更专业一点，最好再分个“/home”分区。这里以后存的都是你自己的各种文件，音乐、电影、图片、各种文档，以

及各种软件的配置文件都在这里，所以要尽量大一点。并且以后如果要重装 Ubuntu 系统，只要保留这个分区不格式化，并且依旧挂载为“/home”，那么这些个人信息就都还在。把上面说的这些都建好了之后，就可以看到类似图 2.17 所示的效果。



图 2.15 创建 root 分区



图 2.16 创建交换空间



图 2.17 手动分区最终效果

我遇到的这个用户似乎有些经验，对分区这种事情比较了解，直接就选择了手动分区，然后分了 20 GB 给“/”分区，又分了 220 GB 给“/home”，还分了 2 GB 的 swap 区（就是交换空间啦）。分的时候我注意到，硬盘里另外的几个分区中似乎住着一个 Windows 7 系统，嗯，看来我有邻居了。

2.3.5 第 5 步：填写一些基本信息

提示：如果上面选了手动分区就是第 6 步，以下同理。

分区完成，进入第 5 步，就会看到如图 2.18 所示的界面。这里有如下几项要填写。



图 2.18　创建用户界面

（1）名字，就是在文本框里写上你的名字呗，遗憾的是不能用中文。我这个用户填了名字叫 lanwoniu。

（2）登录名，就是一般所说的用户名。刚才那个名字是用来显示的，这个名字是用来登录的，以后让你填这个系统的用户名时，填的就是这个。一般这两个名字都一样，于是在用户填写名字的时候，我就替他把登录名也写成 lanwoniu 了，他也没反对，就这样了。

（3）然后是密码，按照国际惯例，输两遍。

（4）计算机名，随便起就行，我的用户给他的电脑起了个名字叫 snail-computer，看来是嫌他的电脑太快了。

（5）最下面还有 3 个单选按钮，我逐个解释一下。

- "自动登录"——这个选项就是说系统启动的时候，自动用你现在创建的这个用户登录，不用输入密码。
- "登录时需要密码"——这个选项就是普通的登录模式，登录时需要选择用户并且输入密码。
- "登录时需要密码并且加密我的主目录"——这个选项在使用上跟第 2 个没有区别，但是这个用户的主目录会被加密存储。

都填好了之后，还是单击"前进"按钮，就进入下一步了。

这里要说明一下。这一步创建出来的这个用户是拥有管理员权限的用户，但是不是 root 用户哦，所以这里不要试图创建 root 用户。可能有的同学听说过 Linux 下面有个 root 用户很好很强大，不过在我们 Ubuntu 系统里，你可以渐渐淡忘这个 root 用户了。这一步创建的这个用户虽然不是 root，但是，这个用户却有着变身成 root 的权力！

提示： root 用户的权力过大，使用 root 登录时如果出现误操作，很容易造成不可补救的后果，因此 Ubuntu 系统默认禁用了 root 用户。

2.3.6　第 6 步：导入用户信息

如果电脑里已经有了其他的操作系统，我可以帮助用户把原来放在那个操作系统上的

一些配置信息，数据什么的导入到新的系统上来。包括原来的浏览器里的书签，原来桌面的壁纸、用户存的图片、文档、音乐等，我都可以顺手给存在我这边。图 2.19 所示是一个导入用户信息的示例。



图 2.19　导入用户信息

2.3.7　第 7 步：确认信息

前面的操作都做完了之后，这里会让你确认一下之前步骤中所做出的各种选择，如图 2.20 所示。尤其是对硬盘分区的修改，要是现在反悔还来得及，因为到目前为止我还没有做任何实质的改动。如果没什么问题，看见没有，右下角那个按钮不是“前进”了，变成“安装”按钮了。别犹豫，来吧！



图 2.20　安装前的最后确认

安装的过程中我会去网上查找有没有可用的更新，如果有什么软件有新的版本了，就不给你装光盘上的，直接从网上下载最新的装上。还有一些光盘上没有的语言包，也会从网络上下载。不过下载的速度一般会比较慢，因为我只会去国外的官方网站上找（我刚出生嘛……就从那来的，所以只认识那）。要是你等不及的话（一般人都等不及），安装的时候干脆把网线拔了，断了我这念想，装得就快了，有半小时也就装完了。当然，这还得看你的电脑速度。

2.3.8　扩展阅读：Linux 中的最高权限

在安装的过程中，咱们介绍了 Linux 系统中有个 root 用户，拥有着最高的操作权限。有的同学可能会说："我知道，root 就相当于 Windows 系统里的 administrator 嘛，都有着最高的权限。"很好，领悟得很快，但是——并不准确。

【并非至高无上的 administrator】

Windows 系统下权力最高的是谁？是 administrator 吗？很遗憾，不是。是 SYSTEM！也就是系统自己，Windows 7 自己。任何管理员的权利都不能大于 Windows 7 自己的权利。你可以试试去把 C:\WINDOWS 下的 regedit.exe 删了。能吗？"哇！我删了耶，没报错。"别着急，刷新几下看看，是不是又出来了？Windows 7 会保护自己，不叫人类破坏。这个初衷看似还是好的，但是当 Windows 7 自己中毒的时候，就不一样了。当他中毒时，就像被外星生命寄生了的人类（异型看过吧？），就不再是正常的人了，不正常的 Windows 7 仍然会努力保护自己，不让人动他身上的任何部分——包括已经中毒变坏的部分。

【真正至高无上的 root】

那么 Ubuntu 下（其他 Linux 系统也是一样）权力最高的是谁？毫无疑问是 root！是这个用来给人类登录的用户。root 在系统中拥有真正的至高无上的权力。他真的无所不能，他可以运行 rm * -rf（危险动作，切勿尝试，后果自负）删除系统中的所有文件。或许我会语重心长地警告他：这么干很危险滴，这么干就都删光光了，这么干我这个系统就嗝屁了，不存在了！但是，当他确认地告诉我，他现在很清醒很冷静，知道自己在干什么之后，我会义无反顾地流着两行热泪按照他的命令去做！哪怕他要格掉整个硬盘，我也照办。这真是，君叫臣死，臣不得不死；他叫我格，我不能不格（Windows 下是不可能在系统运行的时候格掉系统盘的）。

【理念不同带来的权力不同】

会有这样的区别，原因还是我们两个系统的理念不同。

Windows 7 认为：人类是会犯错误的，很可能一不小心就把系统搞坏了，所以必须加以限制。有些事情让做，有些事情无论如何不能让他们做。而我总觉得，人类是聪明的，他们知道自己在干什么——尤其是用 root 登录进来的人。我认为他是了解我，了解整个电脑才会用 root 登录进来做事情的。所以他的命令不会受到任何的阻挠。而一般的用户会用普通账号登录，既然用普通账号登录，就说明他们承认自己只是个使用者，可能会做错事。那么我就会稍微进行限制，让他们不会破坏我，也不会破坏其他用户的东西。所以，当你用 root 账户登录进来的时候，一定不要辜负我对你的信任。

2.3.9 扩展阅读：Linux 的分区和挂载

在安装的过程中，有的同学对这个 Linux 系统的分区还是不大明白，因为跟 Windows 下的有些区别。没事，咱们仔细说说。

【你们住房子，我们住硬盘】

话说我们软件要想在一台电脑里定居，得有个住的地方，就好像你们人类要在一个城市里定居得有个住处一样。不过我们软件并不像人类一样，住在钢筋水泥的格子里面，我们住的地方，是一块叫做硬盘的空间。说起来我们住的这个硬盘空间，和你们人类住的这个房子是很相似的。那我们就拿您这房子来做对比，说说我们这个硬盘空间吧。

首先，你们人类的房子就是一大块能放东西的空间，是吧。有大有小，100 平米的、200 平米的、40 平米的都有。里面放着洗衣机啊、电冰箱啊、床啊、桌子啊之类的各种东西。我们软件住的硬盘也是一大块能放东西的空间，大小也不一定，什么 80GB 的、200GB 的、500GB 的、一个 TB 的都有。里面存着文档、电影，各种程序，以及我这个操作系统等数据。

你们的房子一般不会是一个整个的空间（毕竟那是要住人的，不是仓库），而是会被分割成几个小的空间，一间屋，一间屋的。我们的硬盘虽然也可以好几百 GB 整个用，可也不是很方便，一般也会被分割成几个小的空间，每个空间就叫一个分区，一个分区就好比你们那一间屋。

【住房要有很多功能空间】

好，关于屋子和分区的事情暂时先放放，说说你们人类日常生活习性的问题。你们一般每天要吃 3 次饭，一般不愿意露天吃，是吧，需要有一个吃饭的地方。而且既然吃饭，就得有个做饭的地方，甭管是谁做，反正得加工一下，不像兔子似的路边上逮着块草地就能过去啃两口。一天 3 顿饭之后，得休息，需要有睡觉的地方，大桥底下也好，水床上面也罢，总得有个地方。那么刚才说的你那房子里，就有为满足你的各种需求而设计的各种功能空间。有放着床睡觉的地方；有摆着炉灶，锅碗瓢盆啥的地方，那是做饭的地儿；放个饭桌，这一看就知道，吃饭的地儿；放个马桶，那这就是厕所。

那么我们 Linux 的硬盘里也有类似的情况（当然，我可不是说我们这也有厨房厕所啊）。我们 Linux 系统有着独特的目录结构，最基本的是一个根目录，我们喜欢叫它“/”，它就像您那整个一大间屋子。“/”目录下还有很多的目录，比如“/etc”，是用来存配置文件的；“/bin”是用来放二进制程序的；“/boot”是用来放启动文件的；“/lib”是用来放库文件的；还有“/home”是用来放用户的各种文件的。这一个个的目录，就好像你房里一个个的功能空间一样，各有各的用途。

【分区和目录的联系】

那么说了这么半天，又是分区，又是目录的，分区跟目录有什么关系呢？有人说了：“我知道，分区就是 C 盘，D 盘，E 盘这些，每个盘里再有各自的目录。”兄弟，我只能告诉你，你又 OUT 了。刚才我说了我们 Linux 的目录结构，就是一个“/”目录，下面有一些次级目录，每个次级目录下面再有子目录及子子目录……无论分区情况如何，这个目录结构是不会变的。那么分区怎么跟目录联系起来呢？联系就是，你可以指定任意一个目录

里的东西存在某个分区里，如果不指定，则这个目录里的东西存在上一级目录所在分区中，如果上一级目录也没有特殊指定分区，则再上溯一级目录，依此类推。这么一直上溯，就一定会上溯到最上层的根目录“/”。所以，装系统的时候，其他的可以不指定，但一定要指定“/”目录存放在哪个分区。

比如说，可以整个硬盘就一个分区，然后指定根目录“/”存在这个分区中。好，那么整个“/”目录，以及“/”目录下的各级子目录里面的所有东西，都存放在这个大分区里。我也可以分两个区，分区甲和分区乙。我指定“/”目录存在分区甲里面，然后指定“/home”目录存在分区乙里面。那么整个“/”目录，以及“/”目录下的，除了“/home”目录及其下各级子目录外，其他目录里面的所有东西，都存在分区甲。“/home”目录及其下各级子目录里的东西，存在分区乙。当然，也可以分 80 多个分区，给每一个目录都手动指定一个分区来存放东西——如果你吃得有点多的话。

这种分区和目录的关系，就像你房子里的房间和功能区之间的关系一样。可以为做饭的地方单独分出一间屋子来，叫做厨房。但是也可以是开放式厨房，厨房并不单独放在一间屋子中，而是和饭厅公用一间屋子。同理，可以为“/home”单独指定一个分区，但也可以不单独指定，而是存在“/”所在的分区中，作为“/”下的一级子目录，和“/”公用一个分区空间。是不是很像呢？

最后再说一点，给某一个目录指定分区的动作，有个专业术语，叫做“挂载”，以后还会经常提到。

2.4　G 大叔——介绍启动管理器 Grub

经过漫长的等待之后，安装终于完成了。我总算离开了光盘，在硬盘里落户了。用户随即发出命令：重启！我满怀信心地看着已经来到硬盘上的兄弟们：“我们就要开始一段新的生活了，希望大家能够做出最大的努力，让用户认可我们这个系统。”看着兄弟们意味深长地对我点了点头之后，我静静地，闭上了眼睛，安心地睡去了，等待着 G 大叔把我叫醒。

2.4.1　计算机启动流程

“嘿，小子，起床了！”

我睁开眼，看看眼前站的人，是门房的 G 大叔。我仔细回忆了一下……哦，想起来了。我刚刚被安装到一台电脑里，这是我的第一次启动。

有人说，你记性怎么这么差啊，这才几秒钟前发生的事情，你怎么就忘了？别奇怪，这是我们软件族的特点。一方面，几秒钟对我们软件来说已经是很长的时间了。另一方面，我们软件不像你们人类，睡觉的时候还能做个梦啥的。我们睡觉的时候（也就是系统没启动的时候），是什么也不知道的，之前发生的事情，需要记忆的，我们都会在睡觉前写成文件放在我们住的硬盘里，这样下次起床就能回忆起来了。

【起床的过程】

每当用户需要我起床工作的时候，他就会按下计算机的电源键，然后，就开始了我漫

长的起床过程。

首先，当计算机的电源键被按下时，会有一股温暖而舒适的电流从电源涌入，流遍整个主板，流经每个元件，流到 BIOS 居住的那颗芯片里。BIOS 就是开机时你按 Del 键进去的那个蓝屏幕（不是所有主板都按 Del 键进 BIOS）。BIOS 这个家伙其实也是一个软件，但他是一个特殊的软件，特殊到一般都不归在软件的行列里，而是被叫做“固件”，因为他住在主板上的一个芯片里，而不像我们住在硬盘里。电流流到 BIOS 住的芯片后，会由芯片上的某一根管脚流进芯片内部，并准确无误地击中的 BIOS 的身体，于是——BIOS 就醒了（合着天天被电醒的，真惨）。

提示：BIOS 是 Basic Input-Output System（基本输入/输出系统）的简写。

BIOS 醒来之后就开始工作。他的工作平凡而重要，复杂而机械，就是去检查 CPU、内存、显卡等是否都正常。都检查一遍没有问题之后，就来到我们住的硬盘这里，来到那间传达室，完成他的最后一个任务——叫醒在门房值班的那个人。

我搬到这里之后，门房里值班的人，就是 G 大叔了。G 大叔大名叫做 Grub，现在已经是 2.0 版本了。他是一个启动管理器，平时就住在传达室。所谓传达室，学名叫做 MBR，是一个硬盘的入口，硬盘的第 0 号扇区。传达室不属于任何一个房间，或者说，MBR 不属于任何一个分区。传达室很小，只有 512 Byte。由于传达室地方实在太小，因此 G 大叔会把一些有用的文件放在我的硬盘空间里，必要的时候来看看。

提示：MBR 中的内容主要有两部分，一部分是启动代码，另一部分是硬盘的分区表。由于空间有限，只能写下 4 个分区的信息，因此一块硬盘最多只能有 4 个主分区或扩展分区。

G 大叔被 BIOS 叫起来之后，会来我的硬盘里读取/boot/grub/grub.cfg 文件，根据这个文件的内容来决定他的动作。这个文件里写了启动的时候应该给用户多少个选项，每个选项都是什么，背景啥样，等待多长时间等。G 大叔按照这个文件上的要求显示给用户一个多系统选择的界面，就像图 2.21 这样（Grub 默认界面应为黑底白字，本书为了提高印刷后的图片质量，特做反色处理）。

```
GNU GRUB  version 1.98-1ubuntu7

Ubuntu, Linux 2.6.32-24-generic
Ubuntu, Linux 2.6.32-24-generic (恢复模式)
Memory test (memtest86+)
Memory test (memtest86+, serial console 115200)
Microsoft Windows XP Professional (on /dev/sda1)

Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the commands
before booting or 'c' for a command-line.
```

图 2.21　Grub 启动界面

用户做出选择后，如果是选择了我，那么 G 大叔就像刚刚这样，来到我屋里，叫我起床，于是我这个 Ubuntu 系统就启动了。

2.4.2　多系统的共存

可能有同学会问，那 G 大叔除了叫你之外，还能去叫谁呢？G 大叔是个启动管理器，如果只负责引导我一个系统，那还怎么称得上“管理”二字呢？G 大叔能够支持多种系统的引导。比如那个水果公司的 Mac OS 系统、与我们 Linux 同宗的 BSD 系统、微软公司的“查皮”Windows XP 系统、“喂死它”Vista 系统、“温妻”Windows 7 系统，甚至“剁死”（DOS）系统等，G 大叔都能够支持。

具体怎样才能让 G 大叔引导其他系统呢？这个不用您操心，我们 Ubuntu 系统在安装的时候，会派 G 大叔去检查硬盘上已经存在的系统，然后根据情况制定出合理的计划。这一切都是 G 大叔自动完成的，您装完系统重启计算机后，就能看到硬盘上已有的系统已经被 G 大叔正确识别出来了。

提示：Max OS 需要安装在主分区才能够正确地被 Grub 引导。

就拿我定居的这台电脑做例子吧。

我们来之前，电脑里已经住进了一个操作系统，就是微软公司的 Windows 7。Windows 7 一个人住的时候，传达室是没人的，只放了一个简单的类似门铃的装置。BIOS 来传达室叫人的时候，只要按一下那个铃，那边的启动管理器就会去叫 Windows 7 起床了。

G 大叔搬进去的时候，会考虑到原有的 Windows 7 系统，查看一下叫醒 Windows 7 的那个门铃连到了哪里。一般是连到一个叫做 Bootmgr.exe 的程序，一按门铃，Bootmgr.exe 就被叫醒了。于是 G 大叔记好 Bootmgr 的位置，再记录好我的位置，生成一个 grub.cfg 文件，放到/boot/grub/目录下。咱不是说过么，我们软件要想记住点什么东西，都得写成文件放到硬盘里，G 大叔也不例外。

当用户启动电脑，G 大叔被叫醒的时候，他就会一脸严肃地问用户：要用哪个系统？一个 Ubuntu 一个 Windows 7，给你 10 秒，快选！如果用户选我，G 大叔就来叫醒我，如果选 Windows 7，G 大叔就去按照之前记录的位置，找到 Bootmge，一脚把他踹醒。

虽然 G 大叔说话有点不客气，不过工作还是尽职尽责的，多数常见的操作系统、常用的分区格式，G 大叔都不在话下。他总说，传达室不是某一个系统专用的传达室，在传达室工作的软件，就该为硬盘上的每一个系统都服务好。

提示：Windows XP 之前的 NT 内核系统使用的是 ntldr 作为启动管理器，从 Vista 开始换成了 Bootmgr。新版的 Grub 对这两者都可以完美支持。

2.4.3　重装 Windows 后 Grub 的修复

可是 Windows 7 那边的作风就不一样了。

如果硬盘上已经住进了我，门房里已经有了 G 大叔，这时候重装或者新装 Windows 7 系统的话，Windows 7 就不管三七二十一地把 G 大叔赶出来，在传达室装好他的“起床铃”

就走了，不管我这边的情况。电脑再启动的时候，BIOS 就找不到 G 大叔，只能去按那个铃，直接启动 Windows 7，我的存在就完全被无视了。

如果这样的惨剧不幸发生了怎么办呢？没关系，他能把 G 大叔赶出来，我照样能让 G 大叔再搬进去！想强拆？没门！不过具体怎么操作呢？再重装一遍 Ubuntu 系统？不用。还记得那张安装光盘么？还记得我说系统出问题的时候可以用它来修复么？没错，就是那张，赶紧让你家狗狗把他叼回来，现在用上了！

像安装的时候一样，用 LiveCD 启动电脑，选择试用，这样就启动了光盘上的 Ubuntu 系统。这时候，电脑可就归我们 Linux 管啦！嘿嘿，小小的 Windows 7 算什么，你想把我们的 G 大叔撵走就霸占整个硬盘了？想得美！LiveCD 启动之后，打开命令行，运行 sudo -i，获取权限。然后运行：

```
$mount /dev/sdax /media/
```

提示：$符号是普通用户的命令提示符，不是命令的一部分，不需输入。

这里 sdax 就是你安装 Ubuntu 的时候用作根目录“/”的那个分区，如果你还单独分了“/boot”分区，那么还得运行：

```
$mount /dev/sday /media/boot/
```

当然，这里的 sdax，sday 都需要根据你的实际分区情况修改，可能是 sda1，sda4，或者 sdb2，sdc8，都没准儿。mount 好了之后，运行：

```
$grub-install -root-directory=/media/ /dev/sda
```

运行完了就好了。最后重启电脑，熟悉的 G 大叔又回来了。

当然，以上说的都是以后可能发生的情况，目前在我这里还没有这样的事情，隔壁那个 Windows 7 睡得像死猪一样，不会有什么举动的。而 G 大叔早在安装的时候就自动设置好了多重系统启动，刚刚就是用户告诉 G 大叔来叫醒我去干活的。

2.4.4 Grub 的简单配置

起床之后，用户似乎对 G 大叔的举动不是很满意，打算要修改一下 G 大叔的配置文件。

刚才我们说了，G 大叔启动的时候会去找/boot/grub/grub.cfg 文件，这里面记录了 G 大叔应该做的一些事情。不过用户要想修改这些设置，可不需要修改这个文件，而是要改/etc/default/grub 文件。这个文件里，简单明了地记录了 G 大叔应该做的一些动作。

提示：旧版 Grub 的配置文件为/boot/grub/menu.lst，新版 Grub 将原有的一个配置文件分为/boot/grub/grub.cfg 和/etc/default/grub 两个。前者更加复杂，提供给 Grub 读取，以提供更复杂的功能。后者更加简明，提供给用户，用于一些简单的配置。

只见用户下达了命令：

```
$sudo gedit /etc/default/grub
```

这么命令的意思就是，以 root 用户的身份，命令 gedit 软件，去打开/etc/default/grub 文件。输入这个命令之后，我会要求用户再输入一遍他自己的密码，注意，是当前用户的

密码(比如我这里,就是 lanwoniu 这个用户),不是 root 的密码,真正的 root 用户的密码……是个迷。关于这个 sudo，咱们后面还会见到它，这里暂且不表，您只需要知道命令就这么敲就行了。命令运行后，就会看到打开了一个 gedit 软件，里面显示的就是 grub 文件的内容，大约就是这样：

```
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.

GRUB_DEFAULT=0                          #解释 1#
#GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true          #解释 2#
GRUB_TIMEOUT=10                         #解释 3#
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"  #解释 4#
GRUB_CMDLINE_LINUX=""                   #解释 5#

# Uncomment to disable graphical terminal (grub-pc only)
#GRUB_TERMINAL=console

# The resolution used on graphical terminal
# note that you can use only modes which your graphic card supports via VBE
# you can see them in real GRUB with the command `vbeinfo'
#GRUB_GFXMODE=640x480

# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to Linux
#GRUB_DISABLE_LINUX_UUID=true

# Uncomment to disable generation of recovery mode menu entrys
#GRUB_DISABLE_LINUX_RECOVERY="true"

# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
```

这一大堆看着有点乱，不过别害怕。首先，所有以#开头的行都是注释，那是给人类看的，G 大叔会直接忽略掉这些行。剩下的就不多了，好，慢慢介绍。

提示：Linux 下的绝大多数配置文件和脚本都以#作为注释行的开头。

- 解释 1：GRUB_DEFAULT=0 这一行的意思，就是让 G 大叔在用户没有选择的情况下，默认来叫醒我。因为在电脑启动的时候，G 大叔给用户的选项里，叫醒我是排在第 1 个的（但是 G 大叔数数喜欢从 0 开始数，所以是“=0”）。这里也可以写 saved，意思就是记住上一次开机的选择。上次选的谁，这次就默认选谁。
- 解释 2：GRUB_HIDDEN_TIMEOUT_QUIET=true 是说倒计时的过程中不显示秒数，只默默地计时。如果这一行设为 false，则 G 大叔会显示倒计时还剩下多少秒。
- 解释 3：GRUB_TIMEOUT=10 这一行是 G 大叔给用户选择的时间，也就是 10 秒钟。如果不选就根据 GRUB_DEFAULT 的设置，选择默认的系统去了。如果不希望有时间限制，就设置为–1。.
- 解释 4：GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"这一行是启动时 Grub 大叔要传给我这个 Linux 内核的启动参数。这个参数只有在正常启动的时候会传给我，recover 模式就不传了。quiet 的意思是告诉我，启动时不要打印任何信息。splash 是告诉我要显示启动画面。

- 解释 5：GRUB_CMDLINE_LINUX=""也是启动参数，跟上面那一行不一样的是，这一行设置的参数无论是什么启动模式，都会传给我。

我这个用户主要是想改改默认启动的系统，于是把 GRUB_DEFAULT 改成了 4。我掐指一算，改成 4，也就是启动时 G 大叔给用户的第 5 个选项……哦，是去叫醒 Windows 7。哎……看来还是不喜欢我啊。另外 GRUB_TIMEOU 也被用户改成了 5，看来这个用户还是个急性子。

改完了之后，保存好这个文件，但还不算完，光改这个文件是不管用的，G 大叔真正关心的是/boot/grub/grub.cfg 文件。还得运行一下：

```
$sudo update-grub
```

这样，就会根据刚才修改的 grub 文件，自动生成一个给 G 大叔看新的 grub.cfg 文件，这就算修改完了。

提示：在 Linux 系统的终端中输入密码时，密码不会回显。

2.5 更多选择

从安装到第一次启动，我还算比较顺利。可能是我遇到的这个用户运气好，也可能是我运气好，遇到个水平比较高的用户，到底是谁运气好，这是个哲学问题……不过总之，会有一些人安装 Ubuntu 不是那么顺利，或者因为种种原因安装遇到阻碍。不过没关系，没有什么困难是不能克服的，我们 Ubuntu 系统的安装方法多着呢。

2.5.1 基于 Windows 的 wubi 安装

有人说，我就遇到困难了。你刚才讲了这么半天，那么多步骤，还得分区，还得设什么 BIOS，太复杂了，听不懂啊。有简单点的办法没有？我告诉你，有。

为了使更多已经装了 Windows 7 之类系统的人能够更加简单地安装和体验我们 Ubuntu 系统，我们的光盘里带了一个软件，叫做 wubi。您可别误会，他可不是个输入法，不要妄想用五笔字型输入“我要装系统！”就能把 Ubuntu 装上。这个 wubi 是 Windows UBuntu Installer 的缩写。这家伙是运行在 Windows 7 系统下的软件，他的功能，就是帮助你在 Windows 7 系统下安装我们 Ubuntu 系统。不用你懂分区，不用你知道挂载，不用改变当前硬盘的状态，一切全都交给他就好。并且我们这个光盘还设置好了自动运行，光盘放进去就会看到 wubi 运行的界面了，就是图 2.22 所示的这样。

第 1 个选项，就是之前说的光盘安装。选了这个选项之后电脑就会重启，然后从光盘启动（当然还得设好 BIOS），之后就跟我们说的光盘安装没有区别了，不选这个（不选你说那么半天！）。看第 2 个选项，“在 Windows 中安装”，这个看着新鲜吧？好，就是它了！

单击“在 Windows 中安装”按钮之后，就看到一个如图 2.23 所示的设置窗口。

要设置的东西不多，也都挺简单的，咱们一个一个说。

（1）目标驱动器，就是让你选择把 Ubuntu 装在哪个盘上。一定要找个空闲空间大的

盘，因为装的时候要在那个盘上创建一个巨大的文件，文件的大小就是下面那个“安装大小”里选的大小。这个巨大的文件会被 Ubuntu 当作硬盘来用，系统就装在这里面，因此就不需要调整你的实体硬盘了，避免出现数据丢失。怎么样，很人性吧？

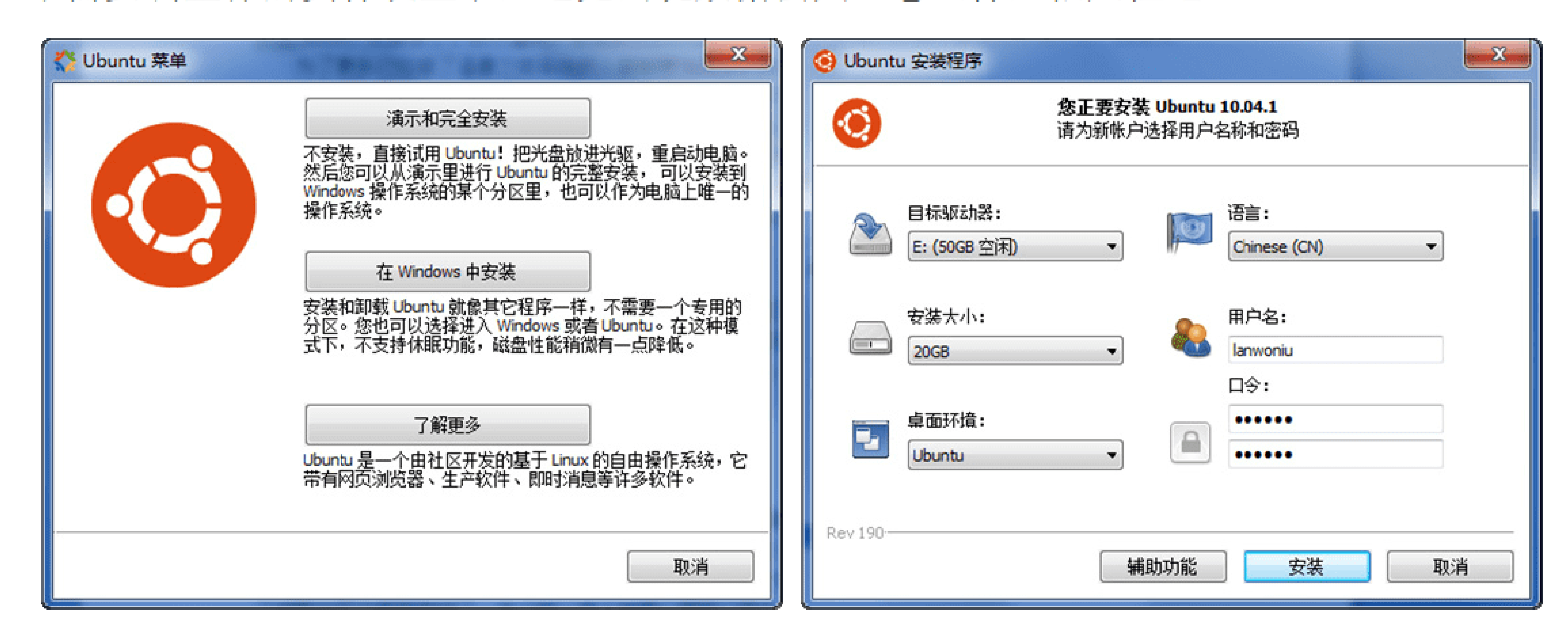


图 2.22　wubi 运行界面　　　　图 2.23　wubi 安装设置窗口

提示：由于 FAT32 支持的文件大小有限，因此选择的目标驱动器需要使用 NTFS 文件系统。

（2）安装大小，不多说了，就是用来当硬盘的那个文件的大小。这个大小一旦确定，装好 Ubuntu 系统之后，可就不能改了，所以一定要想好。如果想日常使用，至少要 20 GB，如果只是装来看看，10 GB 就够了。

（3）桌面环境，没啥可选的，就是 Ubuntu。

（4）语言，你说呢？

（5）用户名，就是安装时创建的那个有变身能力的用户。

（6）口令，就是密码，你知道的，国际惯例。

都选好了，自然就单击下面的“安装”按钮。之后就进入安装的第一阶段，这里没什么可说的，都是毫无悬念的进度条。这个过程结束之后会问你要不要重启。如果你正跟小妹妹聊得火热，待会儿再重启也不妨，如果没什么事情，那就赶快重启看看吧。

重启之后会看到系统选择的界面，装过多个 Windows 的同学会很熟悉，就像图 2.24 所示的这样（此界面应为黑底白字，本书为了提高印刷后的图片质量，特做反色处理）。

选哪个？还用问吗，自然是 Ubuntu。选择之后就进入安装的第 2 个阶段，第 2 阶段也只是没有悬念的进度条而已，等着就好了。装好了再重启，OK，可以进入 Ubuntu 了。

提示：wubi 安装的 Ubuntu 系统由于使用的是虚拟的硬盘，因此磁盘读写的性能要比装在真实硬盘上的 Ubuntu 差一些，并且可能会导致系统不稳定。

2.5.2　U 盘安装

又有人说了，我还有困难。你上面说的方法都得用光盘吧，可我的电脑没有光驱，这可就没法装了吧。不管是申请来的光盘还是自己把 ISO 刻录成光盘都得有光驱呀。别急，

光盘没有，U 盘有没有？有 U 盘就行！

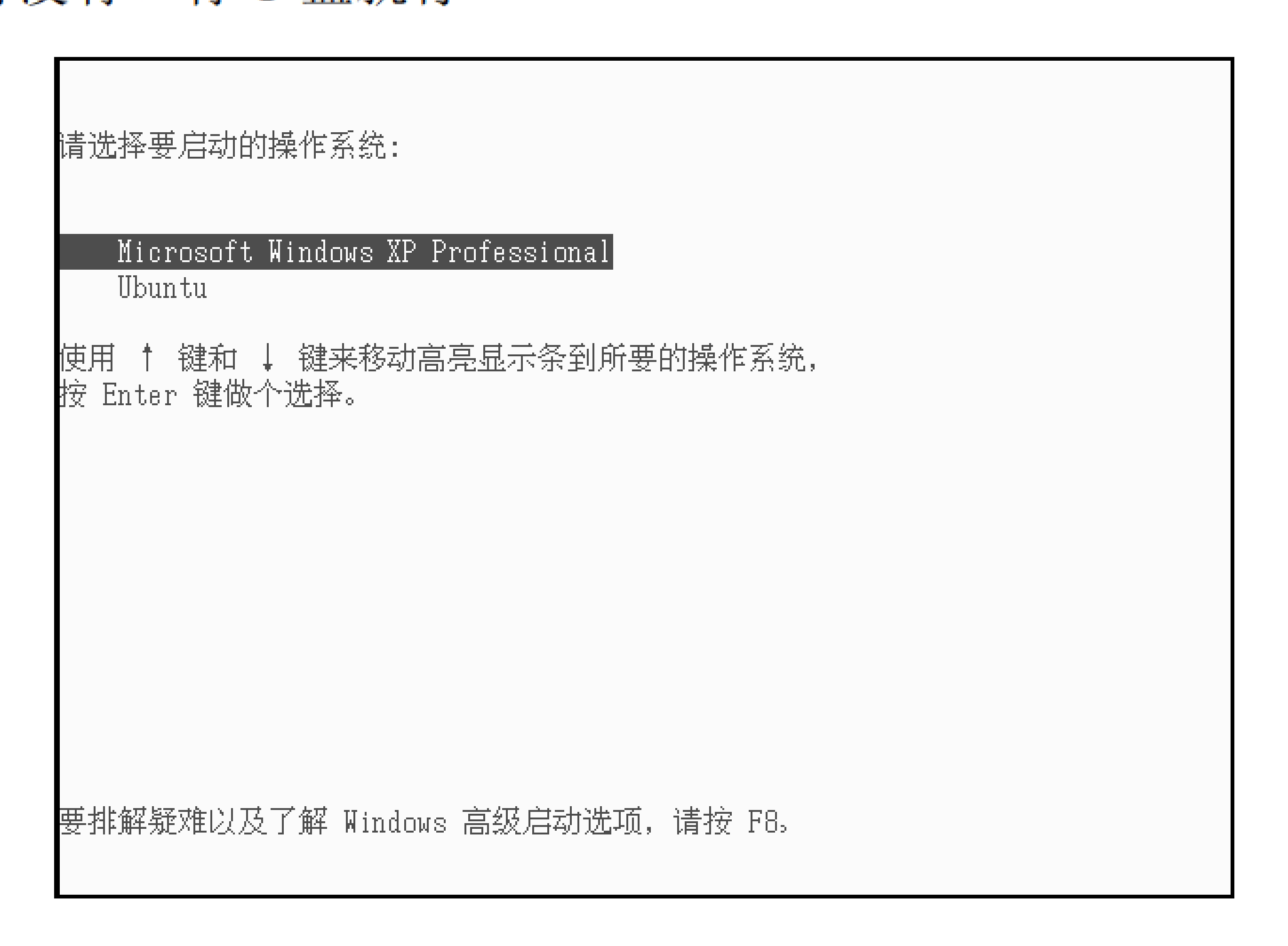


图 2.24　wubi 安装后的多系统选择界面

要想用 U 盘安装，需要几个前提。

（1）你得有个 U 盘（废话！），注意一定得是 U 盘，别拿个 MP3 或者手机糊弄我，弄坏了我可不负责。

（2）你的电脑要支持 U 盘启动，支持不支持查你的主板说明书去。基本上只要你的主板不是那种能在古董市场看到的型号，都应该支持。

（3）就是我们接下来要介绍的，需要一个制作安装 U 盘的软件。

【用 UltraISO 制作安装 U 盘】

话说有这么个软件，叫做 UltraISO。这家伙本事挺大，可以用来刻录光盘。“我没光驱啊，刻录的哪门子光盘啊！”您别着急，我还没说完呢，他不但可以刻录光盘，还能刻录 U 盘。

把您的 U 盘准备好，里面的内容赶紧先找别的地方存起来，一会儿就啥都没了。把 U 盘插在电脑上，运行 UltraISO 软件，会看到如图 2.25 所示的界面。单击左上角的“文件”菜单，选择打开，然后找到你下载的 Ubuntu 系统的 ISO 文件。打开 ISO 文件后的界面如图 2.26 所示。

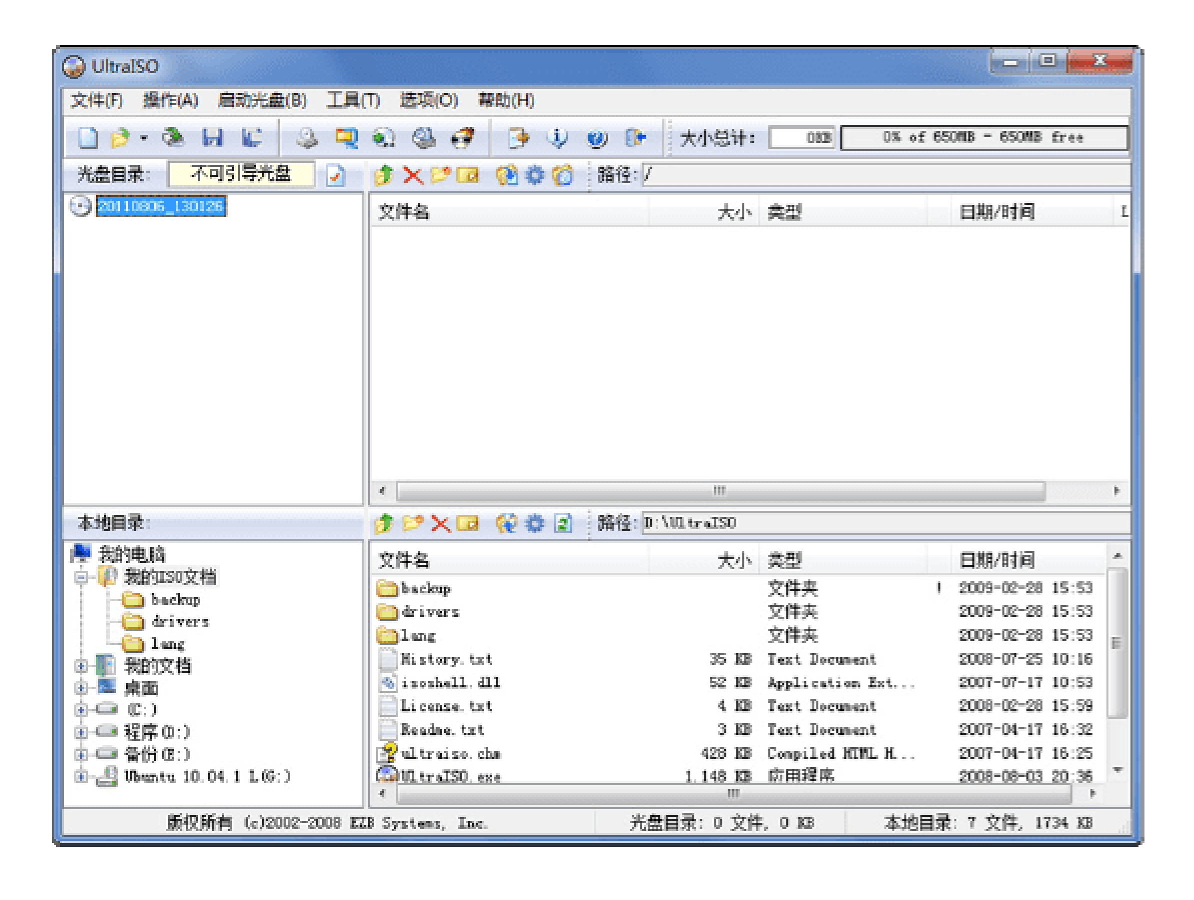


图 2.25　UltraISO 软件主界面

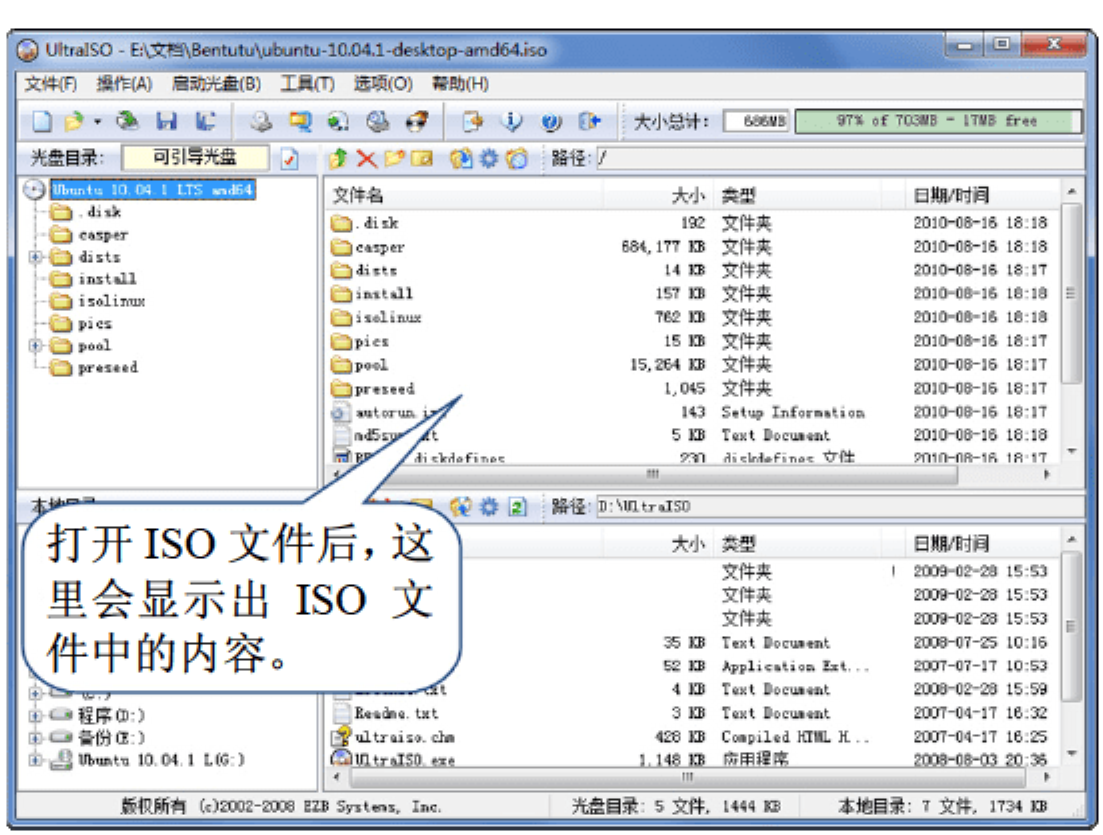


图 2.26　打开 Ubuntu 的 ISO 文件

🔔**提示：**由于 Windows 7 系统提高了系统的安全性，因此在 Windows 7 中运行 UltraISO 软件需要以管理员身份运行。

然后单击“启动光盘”|“写入硬盘映像”菜单，会弹出“写入硬盘映像”窗口，如图 2.27 所示。



图 2.27　“写入硬盘映像”窗口

在该窗口中的“硬盘驱动器”下拉列表框里选择好你的 U 盘，可一定要选对哦，否则就指不定丢多少数据了。“写入方式”选择 USB-HDD+，然后就开始吧，单击“写入”按钮。等写入完了，你的启动 U 盘就制作好了，用它启动电脑就像用 LiveCD 启动电脑一样了。不过要说明的一点，这个 UltraISO 可是要付费的，别偷来就用哦。“我平时也不用，就为了刻录一下还得付费？有点亏啊。”嗯，可能是有点，如果您不想用付费软件，没关系，咱还有办法。

🔔**提示：**此方法仅限于 Ubuntu 10.04 及其以前的系统。Ubuntu 10.10 及其以后的系统，由于 ISO 文件发生变化，无法使用 UltraISO 软件制作安装 U 盘，只可以使用下面介绍的 UNetbootin 软件制作安装 U 盘。

【用 UNetbootin 制作安装 U 盘】

还有个软件，叫 UNetbootin。这个家伙是个免费的开源软件，可以到这个地方来找他：

```
http://unetbootin.sourceforge.net/
```

这个软件同时支持 Windows 7 系统和我们 Linux 系统，您既然是想在 Windows 7 下创建 Ubuntu 的 LiveUSB，那么自然要下载那个“For Windows”版本的。从网上把这个软件下载到你的机器上，不需安装，直接运行。运行之后，就出现图 2.28 所示的界面。

首先，整个界面上有两个单选按钮：一个是“发行版”；一个是“光盘镜像”。

如果选中上面的“发行版”单选按钮，那么软件就会从网络上下载指定的发行版并刻录到你的 U 盘，这属于一站式烧录，不过多数人不这么用。

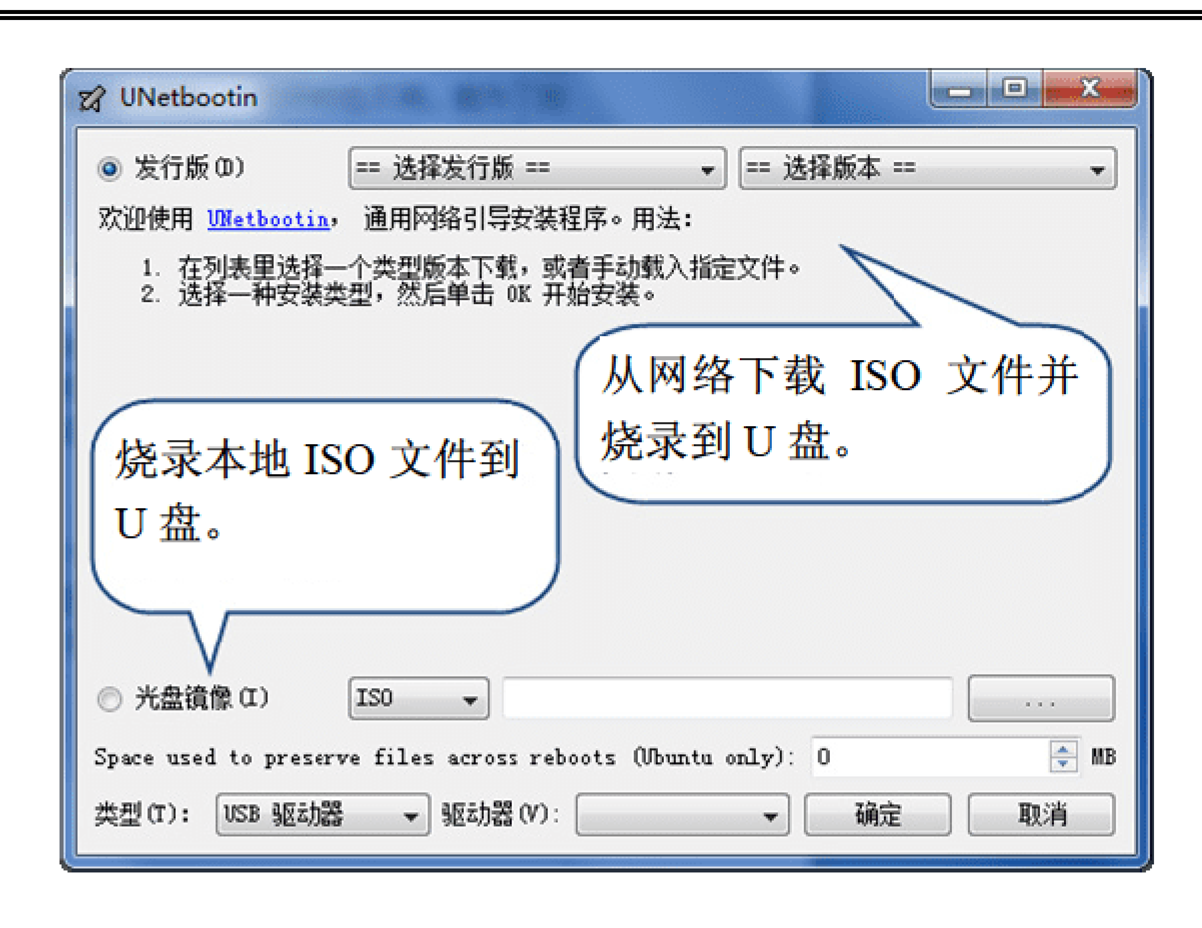


图 2.28　UNetbootin 软件界面

你已经下载了 Ubuntu 的 ISO 文件，所以应该选中“光盘镜像”单选按钮，然后单击后面的“...”按钮（也不给按钮起个好名字），选择存在硬盘上的 ISO 文件。在“类型”下拉列表框里选择“USB 驱动器”，再在“驱动器”下拉列表框中找到对应你 U 盘的驱动器（注意，驱动器一定要选择对！否则的话指不定你哪个硬盘的数据就挂了！）。至于“Space used to……”那一行后面，如果你只是想安装系统，空着就可以了。

确认都选对了之后，单击“确定”按钮，就可以等着了。UNetbootin 理论上是不会破坏你 U 盘上已有的数据的，但是你的 U 盘必须有足够的剩余空间，能放下一个 ISO 文件就差不多了。不过，虽然如此，还是建议你事先备份一下 U 盘的数据。

2.5.3　其他版本的 Ubuntu 介绍

另外，去网上下载 Ubuntu 的同学也许会发现，除了 ubuntu-10.04-desktop-i386.iso 这个 LiveCD 以外，还有很多其他的 Ubuntu 安装光盘，比如什么 ubuntu-10.04-desktop-amd64.iso、ubuntu-10.04-dvd-i386.iso、ubuntu-10.04-dvd-amd64.iso、ubuntu-10.04-alternate-i386.iso 等。这么多种，有什么不一样呢？等我慢慢说给您听。

【i386 和 amd64 的区别】

先说这个，ubuntu-10.04-desktop-amd64.iso。它和 ubuntu-10.04-desktop-i386.iso 唯一的区别就是一个是 i386，一个是 amd64（废话，傻子都知道！）。这里 i386 和 amd64 说的是 CPU 的类型。有的同学会说：“哦，那我知道了，这个 i386，因为有 i 嘛，就是用在 Intel 公司的 CPU 上的，那个 amd64 自然就是用在 AMD 公司的 CPU 上的。”我很高兴地告诉这位同学：“你答错了!”i386 指的是 x86 架构的 32 位 CPU，因为这种架构是在当年 Intel 公司生产 Intel 386 处理器时就确定下来的，所以叫做 i386。之后的奔腾几都是这个架构。还有 AMD 公司，也生产兼容 x86 架构的 CPU，一大堆这个龙那个龙的，都是 i386 兼容的 CPU。后来随着技术的发展，32 位的 CPU 逐渐退出了历史舞台，出现了 64 位的 CPU，至于具体什么是 32 位，什么是 64 位，它们有什么不同，咱们以后会详细说。

最先推出桌面用 64 位 CPU 的，就是 AMD 公司。所以目前普通 PC 用的 64 位 CPU 这种架构是 AMD 公司确定的，于是就叫做 amd64。那什么 CPU 才是 64 位 CPU 呢？基本上你现在能买到的全是！刚才那位同学又说了：“哦，那我明白了。i386 就只能装在 32 位

的古董级 CPU 上，amd64 就只能装在主流的 64 位 CPU 上。”我再次恭喜这位同学——又错了！考虑到现在的很多软件依然不支持 64 位，所以现在的家用 64 位 CPU 都是兼容 32 位的，也就是说在 64 位 CPU 上安装 i386 的系统是可以的，但是要在 32 位 CPU 上安装 amd64 的系统，那确实不行，想都甭想。

【DVD 和 CD 的区别】

然后咱再说说这个 ubuntu-10.04-dvd-i386.iso。最后的这个 i386 不用解释了，跟上面一样。主要就是这个“dvd”。其实这个安装文件跟 desktop 的区别就是，那个是 CD 的，这个是 DVD 的（又跟没说一样）。这个 DVD 的里面比那个 CD 多了一些常用的软件和语言包。不过有一点，默认安装的软件和 CD 版的是一样的，别以为 DVD 的就多给你装什么软件。只不过装完系统之后，可以从光盘安装其他的软件而不用上网去下载。这主要是针对上网不大方便的人用的。有一张 DVD，基本软件就都齐了。另外还有一点好处就是，如果你选择的语言是汉语，DVD 版的装好了之后有比较完整的汉化了，因为 DVD 容量大，可以装下更多的语言包。

【灵活的 Alternate】

再说说这个 ubuntu-10.04-alternate-i386.iso。Alternate 的意思就是安装的时候可以选择安装的软件，可以装成桌面版，也可以装成服务器版，可以有图形界面，也可以没有图形界面。总之，有很多选择，是给高手们预备的。另外这个版本在安装的时候是没有图形界面的，安装时的界面大约类似图 2.29 这样。所以英语好、对 Linux 系统熟悉的同学才可以安装。

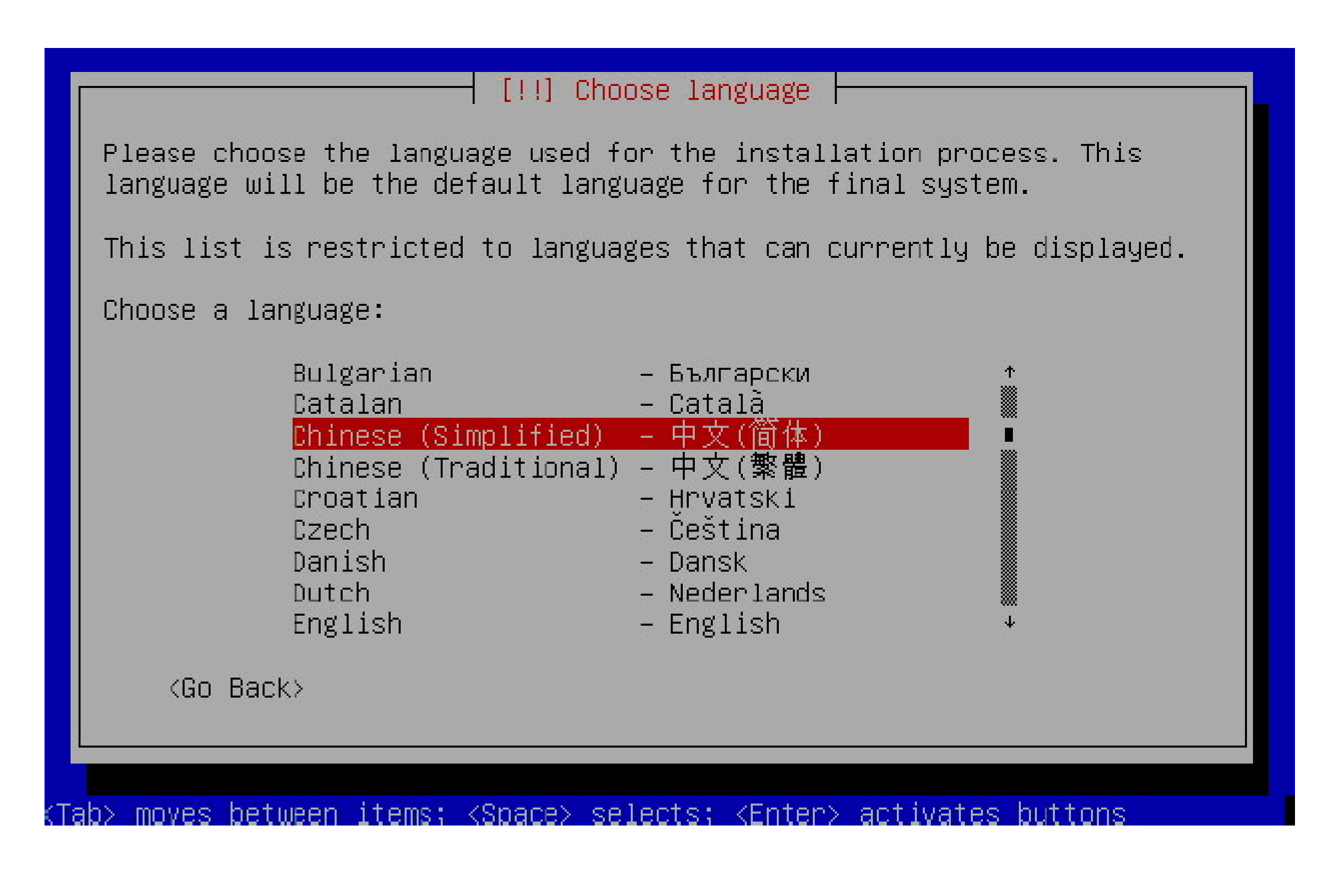


图 2.29　Alternate 版本的安装界面

【高深的 Server 版】

最后，可能有的同学还找到了 Server 版本的 Ubuntu 系统。也许你是打算学习用 Linux 系统搭建各种网络服务器的知识，并且根据以前用 Windows 2003 之类的服务器系统的经验，觉得装个 Server 版就直接从入门到精通了。但是我们 Ubuntu 系统可不是这样，Server 版绝对是专门当 Server 用的，因为——他们完全没有图形界面！

并且这里要说明的是，Server 版和桌面版的区别并不大。一方面是自带的软件不一样，但是 Server 版也可以通过软件源装上桌面版的软件，反之亦然；另一方面，Server 版用的是服务器专用的内核，但同样，桌面版也可以通过软件源，安装服务器版的内核，反之亦然。所以，无论你要用 Ubuntu 做什么，只要你是个新手，就推荐安装桌面版。

【各种其他“兔兔”】

除了我这种标准的“笨兔兔”——Ubuntu 系统之外，我们 Canonical 学校其实还针对不同用户的需求，开设了很多其他的专业，培养出了多种“兔兔”。下面我就给您介绍一下其中最热门的 3 个吧：酷兔兔、小兔兔、育兔兔。

“酷兔兔”，也就是 Kubuntu。他们都是艺术专业培养出来的学生。他们的样子要比我们 Ubuntu 好看些，精致些。这主要是因为负责为他们提供桌面环境的，是 KDE 团队，也因此才叫做 Kubuntu。KDE 桌面环境的特点，就是美观、细腻，并且愿意把各种部分的设置能力交给用户，让用户可以随心所欲地把自己的桌面改成想要的任何样子。图 2.30 所示就是 Kubuntu 系统的一个截图。

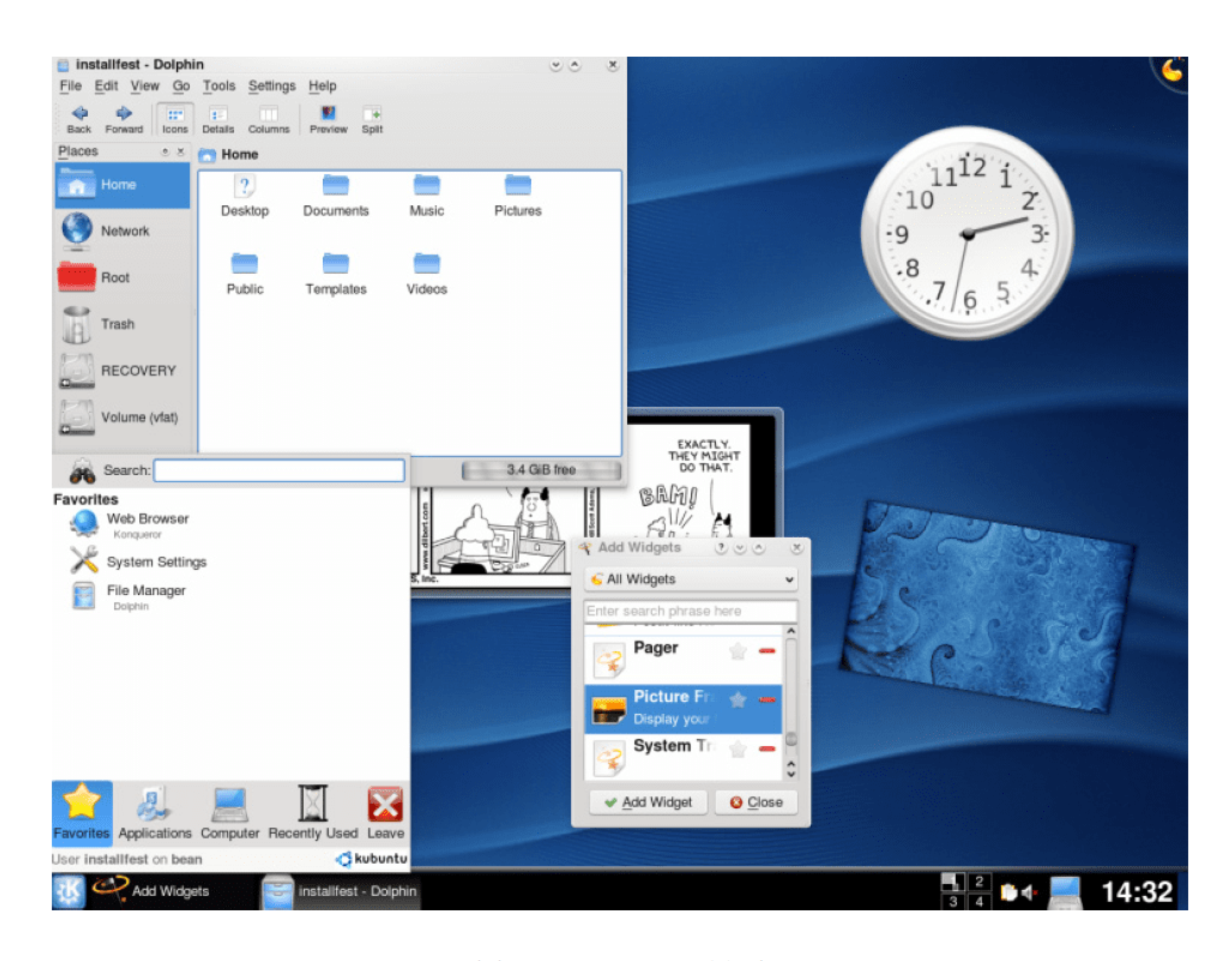


图 2.30　Kubuntu 界面

“小兔兔”大名为 Xubuntu。他们都是准备去艰苦环境下工作的志愿者。Xubuntu 使用的桌面环境不是 Gnome 也不是 KDE，而是 XFCE，样子很朴素，就像图 2.31 所示的那样。XFCE 的特点就是小巧，占用资源少。可以在很艰苦的硬件条件下很好地工作。比如内存，Xubuntu 能够在 200 MB 内存的机器上流畅地运行。当然，相应的软件也要用一些轻量级的。要是在 Xubuntu 下运行一个非常耗资源的程序，那么该慢照样慢。

Edubuntu，我们管他叫“育兔兔”，因为他是教育专业出身。Edubuntu 用的桌面环境跟我一样，只是他附带了很多搞教育的软件，能够教小孩子打字、画画、学习物理知识之类的软件。图 2.32 中展示的，就是一些 Edubuntu 系统里的教育软件。这些软件都是很好的老师，很多小孩子用起来都乐此不疲。很多小游戏也都是寓教于乐的，家长给孩子用这个系统，完全不必担心孩子沉迷于游戏（因为实在没啥可沉迷的游戏……）。

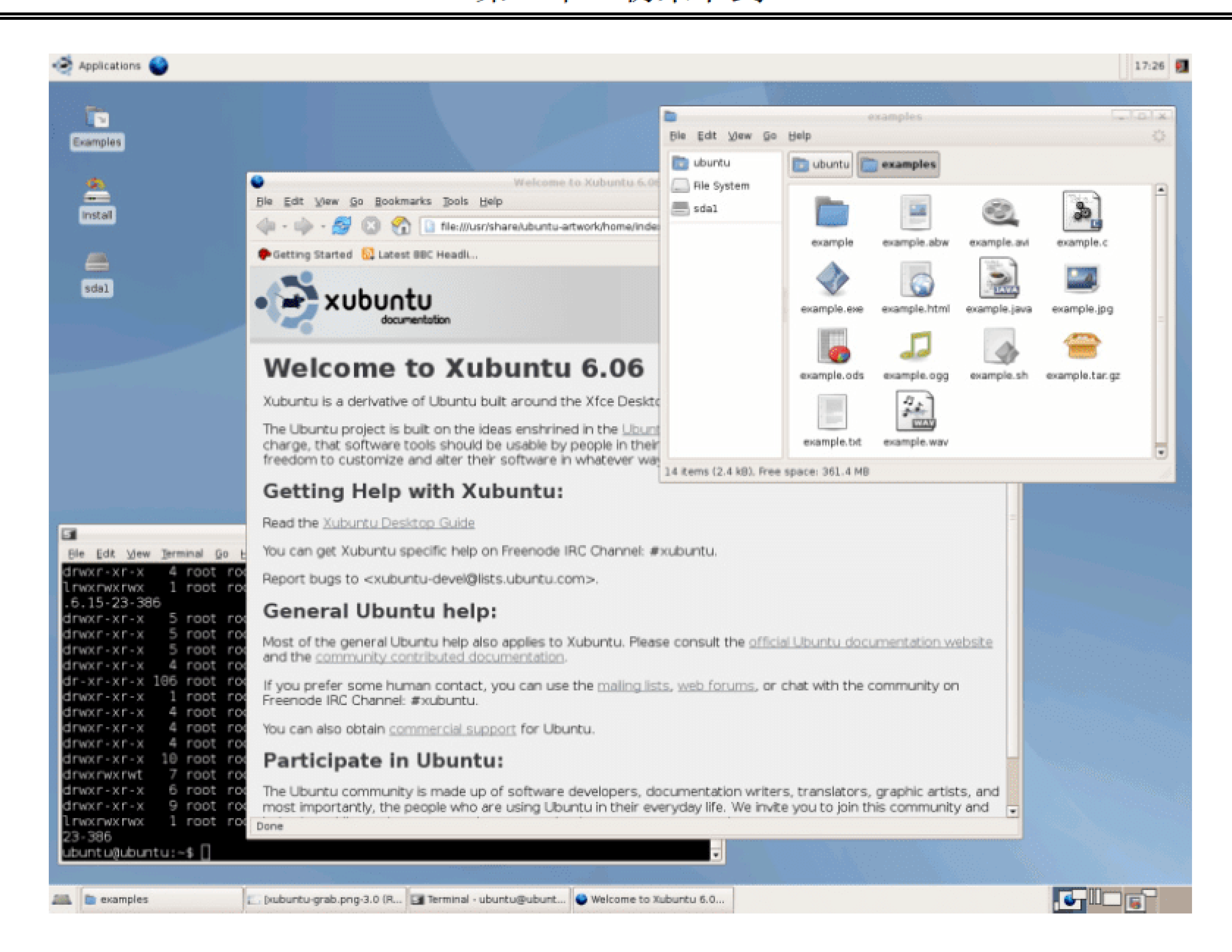


图 2.31　Xubuntu 界面



图 2.32　Edubuntu 中的教育软件

2.6　本 章 小 结

这笨兔子到现在，算是定居到了那位 lanwoniu 用户的 PC 上了。而且生活条件不错，硬盘宽敞，内存也够大，门房的那个 Grub 也幸运地没有遇到隔壁的强拆。折腾了这么半天，也够这兔子累的了。

不过光安装结束还远不算完，后面还有好多事等着他呢。诸位且听下回分解，看用户如何调教笨兔子。

第 3 章　渐 入 佳 境

用户装完系统并改好启动顺序后，马上重启了一下看看效果。于是，BIOS 再一次被电醒，Grub 大叔再一次面无表情地问：要叫哪个？给你 5 秒，不说我就去叫 Windows XP 了啊！用户满意地点点头，然后他选择了启动我这个新装的 Ubuntu 操作系统。于是，刚刚躺下还没把被子捂暖的我，又被 Grub 大叔叫起来干活了。哎，刚安顿下来也不让人家休息一下。不过也是，系统仅仅装好还不算完，还是要做一些基本设置的。

3.1　招贤纳士的 apt

启动完毕后，这位懒蜗牛同学猛然发现，怎么好像菜单都是英文的呢（我终于根据用户输入的用户名，猜测到了他准确的中文名叫“懒蜗牛”，嘿嘿）？装的时候选的是中文啊。是的，虽然选的是中文，但如果用户在安装的时候没有联网，无法下载语言包，界面上就只有少数几个地方被翻译了过来，大部分的菜单还是英文的。那难道就这样凑合着看英文了？当然不是，系统安装好之后再安装中文包也是一样的。那么怎么安装中文包呢？又怎么安装其他需要的软件呢？嘿嘿，很简单。

提示： 如果是 DVD 光盘版的 Ubuntu，就可以在安装时装好完整的中文支持，CD 版因为没有足够空间而无法提供完整的语言包。

3.1.1　不一样的软件安装方式

懒蜗牛同学显然还没明白过来为什么满眼英文，他决定要为自己的记忆力讨个说法。于是他去查看了一下系统的语言设置。我看到他的这个操作，就赶紧很诚实地告诉他：你选的确实是中文，但是呢，由于一些错综复杂的原因（其实主要是因为我曾经逃了那么 7、8 节中文课），相关的一些语言包并没有完全安装，那么你现在要不要马上装呢？就像图 3.1 这样给他提示。懒蜗牛同学自然是义无反顾地单击了 Install 按钮，我一看用户决定要装中文包了，就赶紧去硬盘里叫醒专门负责安装软件的 apt。

提示： 要查看系统语言设置，单击界面上方的 System |Administration |Language Support 菜单，选择的步骤如图 3.2 所示。

【apt 在 Ubuntu 中扮演什么角色呢】

这个家伙在我们 Ubuntu 系统里的角色，就像一个公司里的人事部经理兼后勤部长。系统里装个软件、卸个软件的，都归他管。软件所需要用的各种环境文件啥的，也都是他

负责准备。不像隔壁那个 Windows 7 似的，装个软件要用户自己上网找、自己下载、自己安装。在我们这，apt 全替您搞定了。当别人夸奖他的时候，他总是自信地拍拍自己的胸脯说：“本 apt 有着超级牛力。”因为他老把这句挂载嘴边，所以时间长了，我们就都管他叫“超级牛力”了（以下“超级牛力特指 apt）。不过他也确实挺厉害，很牛很给力，工作起来兢兢业业，对于每个软件的各种情况了如指掌。要招一个软件来的时候，他会做好所有准备工作，这个软件需要用什么样的库，或者需要什么其他的软件协同工作，他都会事先做好准备。要是哪个软件不幸被用户辞退了，他也会帮忙把那个软件相关的所用东西一一清点好，清理出硬盘。

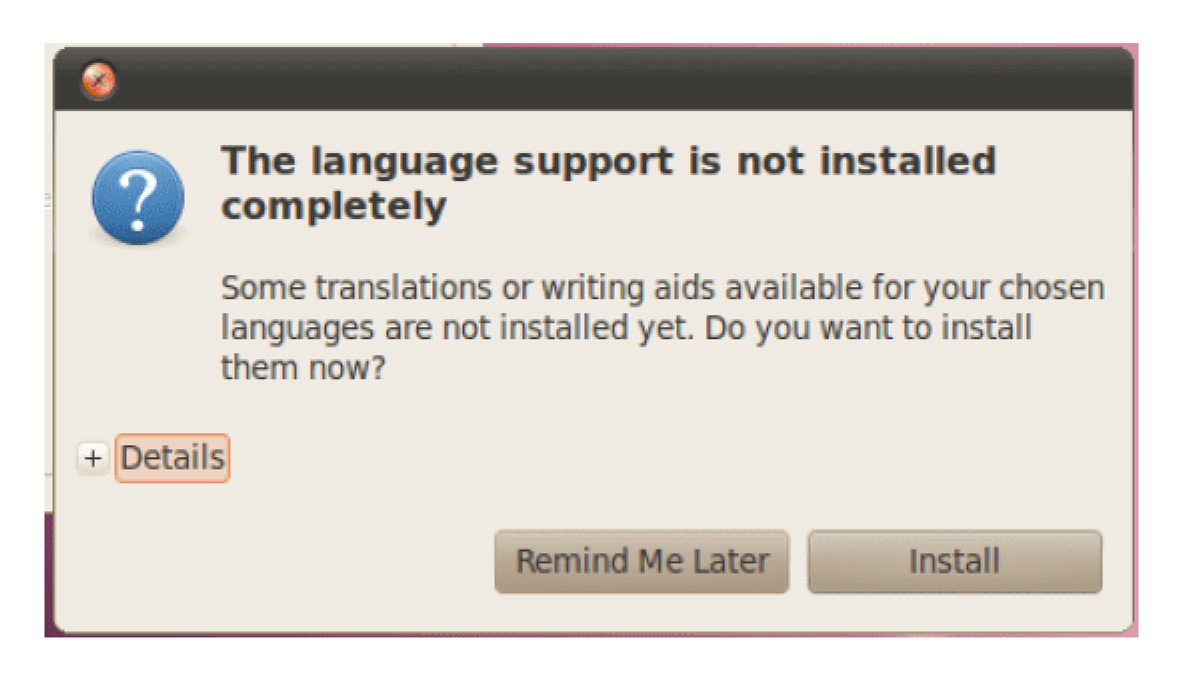


图 3.1　系统提示语言包未完全安装

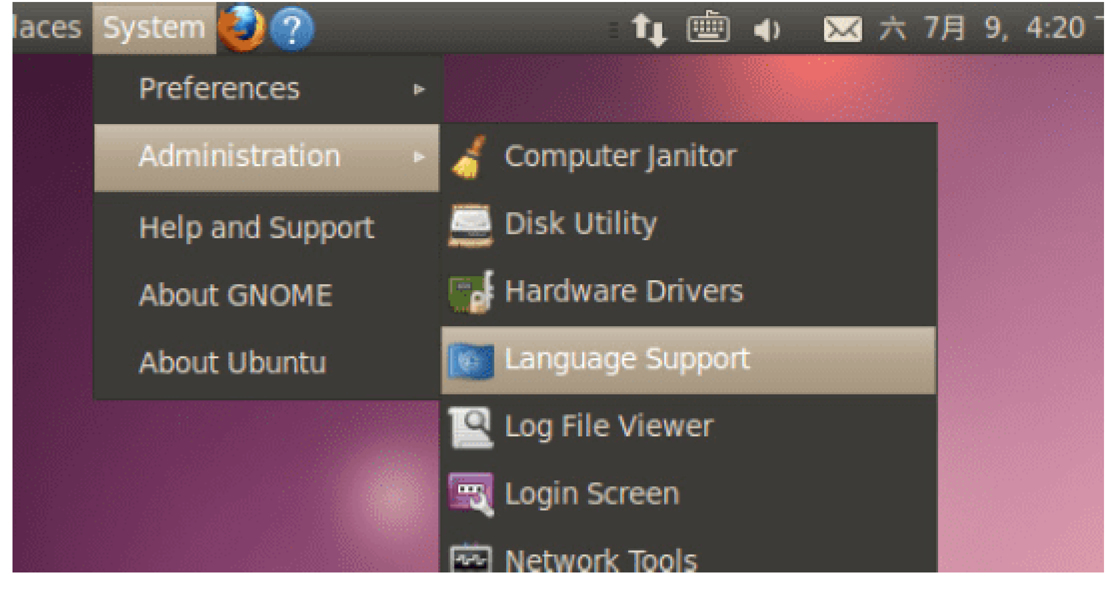


图 3.2　查看系统语言设置

【apt 是如何工作的呢】

比如，用户想用 Vim 文本编辑器来编辑文件，就叫 apt 去招 Vim 来工作。apt 就会报告，说 Vim 要来的话，首先需要准备好 libncurses 这个库和 python 这种脚本语言的执行环境。征得同意以后，他就会去网上找这些东西，并且运回家，把库放在该放的地方，相关的软件安排好住宿。然后他再去找 Vim 同志，请他过来帮忙干活，并且说明：需要的环境都已经布置好了，来了之后马上就开工。每次新人来了之后都很感谢 apt 同志为自己做的这些准备工作，该有的东西，该来的助手都在，于是干活就事半功倍了。但把人才请来之后，apt 同志的工作还没有结束，他还要把现在的人事情况记录下来，方便以后查看。万一哪天用户问起“我说超级牛力啊，咱这现在都有多少软件啊，都是谁啊？”“装的那个某某软件的版本是多少啊？”之类的问题，apt 也能从容地回答。

可以说，apt 这家伙对于我来说实在是非常重要的，有了他，Ubuntu 才是 Ubuntu。所以，在我们这里，要忘记 Windows XP 那种安装软件的方式，装软件就跟超级牛力说，让他去办，省心，放心。

小提示：想看 apt 说自己有超级牛力吗？在终端运行 apt-get --help 试试。

这不，现在懒蜗牛同学说要装中文环境，我就叫来了超级牛力，告诉他去装一些中文字体和中文的输入法。超级牛力听完之后拍拍胸脯说：“放心吧，本 apt 有超级牛力！”说完一转身，从网口跑出去，到网上下载东西去了。

3.1.2　选择合适的软件源

平时超级牛力干活是很麻利的，可是今天我在内存里等了足足有一秒钟，才见超级牛

力拖回来 1 KB 的数据。我赶紧拉住他问："怎么回事？怎么这么慢啊！照这速度得什么时候下载完啊？"他叹了口气："唉，别提了，这路太远了，得翻过 6 个路由，跨过 8 道防火墙，路还窄，不是车多流量大就是行驶缓慢。费好半天劲才拖回这么一点。本 apt 有超级牛力，也架不住堵车啊。"我问他："你是去哪下载的这些东西啊？"超级牛力说："就是那个默认的 http://cn.archive.ubuntu.com/ubuntu/，本 apt 有超级牛力。"唉，这家伙，我知道是怎么回事了，这得从超级牛力的工作原理说起。

【有个东西叫软件源】

超级牛力的职责之一是从网上下载需要的软件，但他去哪下载呢？不是瞎找，而是去一个专门的叫做"软件源"的地方。那里为 Ubuntu 系统提供各种打包好的软件，以及相关的信息介绍。软件源有很多，遍布世界各地，具体该去哪里，是由"软件源列表"决定的。这个列表就写在/etc/apt/sources.list 这个文件里，超级牛力工作的时候，就按这个文件里写的地址去下载软件。现在系统刚刚装上，这个文件里写的是默认的软件源地址。有的人可能离默认的软件源挺近，速度挺快。有的人可能就得绕过半个地球才能过去，那自然就慢了。所以装完系统之后，要根据自己的情况，换一个速度快的软件源。

现在我这位懒蜗牛同学就是没有设置好软件源的典型案例。超级牛力一边下载，一边向懒蜗牛同学汇报进度："这个……根据计算，目前的下载速度大约每秒钟 1 KB，乐观地估计能赶在今年春节前下载完所有语言包。"懒蜗牛同学显然对这个结果很不满意，于是他做出了一个明智的决定——停止下载，先设置软件源！好，那咱们就该说说怎么设置软件源了？

【方法 1，通过软件源管理器】

这是比较简单、省事的方法，推荐新手使用。

（1）单击界面上的 System（系统）|Administration（系统管理）|Software Sources（软件源）菜单，如图 3.3 所示。这就是要启动软件源管理器。



图 3.3　打开软件源管理器

（2）进入这个软件之前需要确认一下您的身份，也就是需要输一下密码，只有管理员级别的用户才能够修改软件源。

（3）输入密码之后就可以看到软件源管理器的主界面了。可以看到下面有个 Download

from，右面有个下拉列表框，现在选择的应该是“Server for 中国”，我们要选那个下拉列表框里的“Other…”选项，如图 3.4 所示。

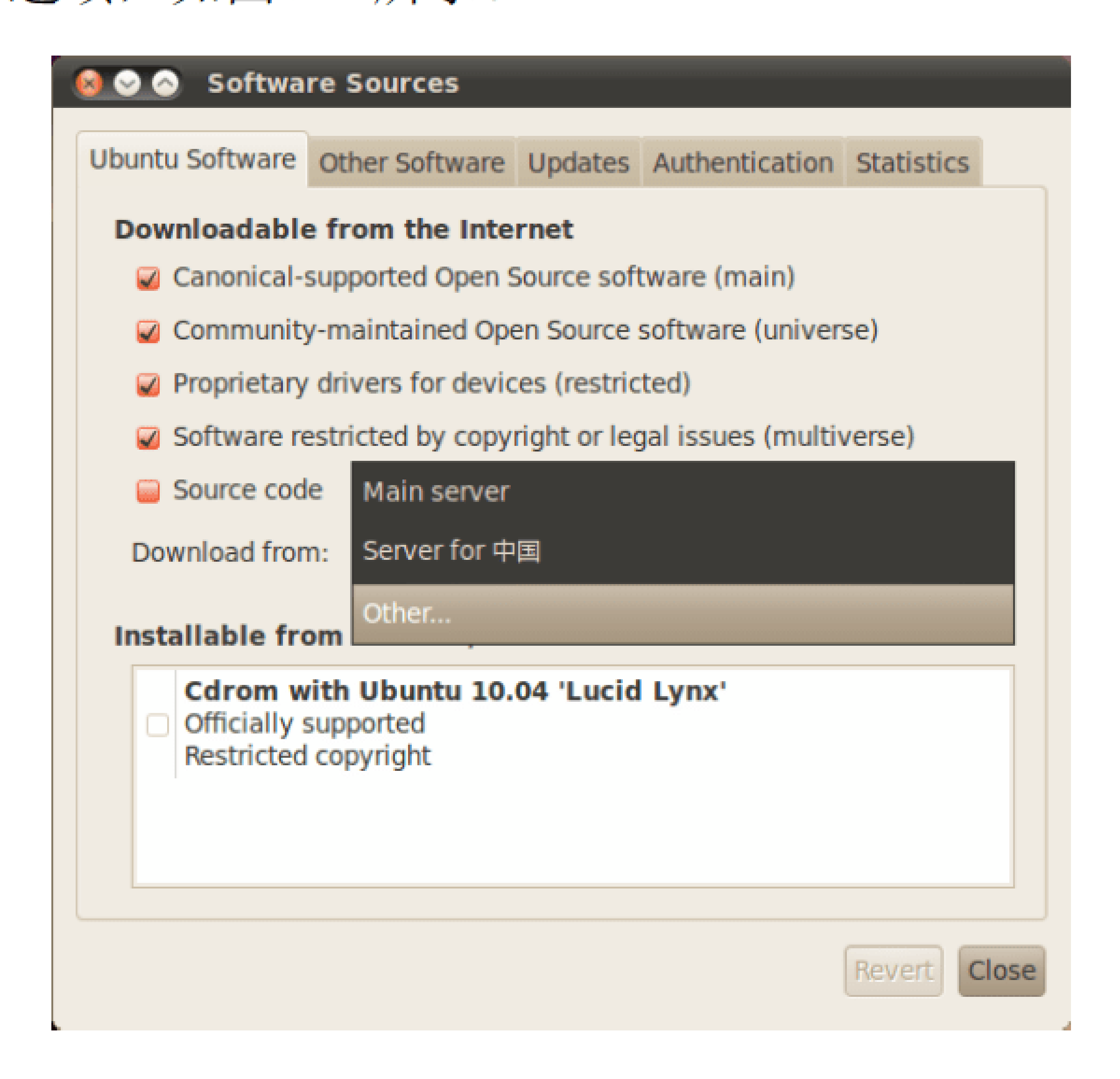
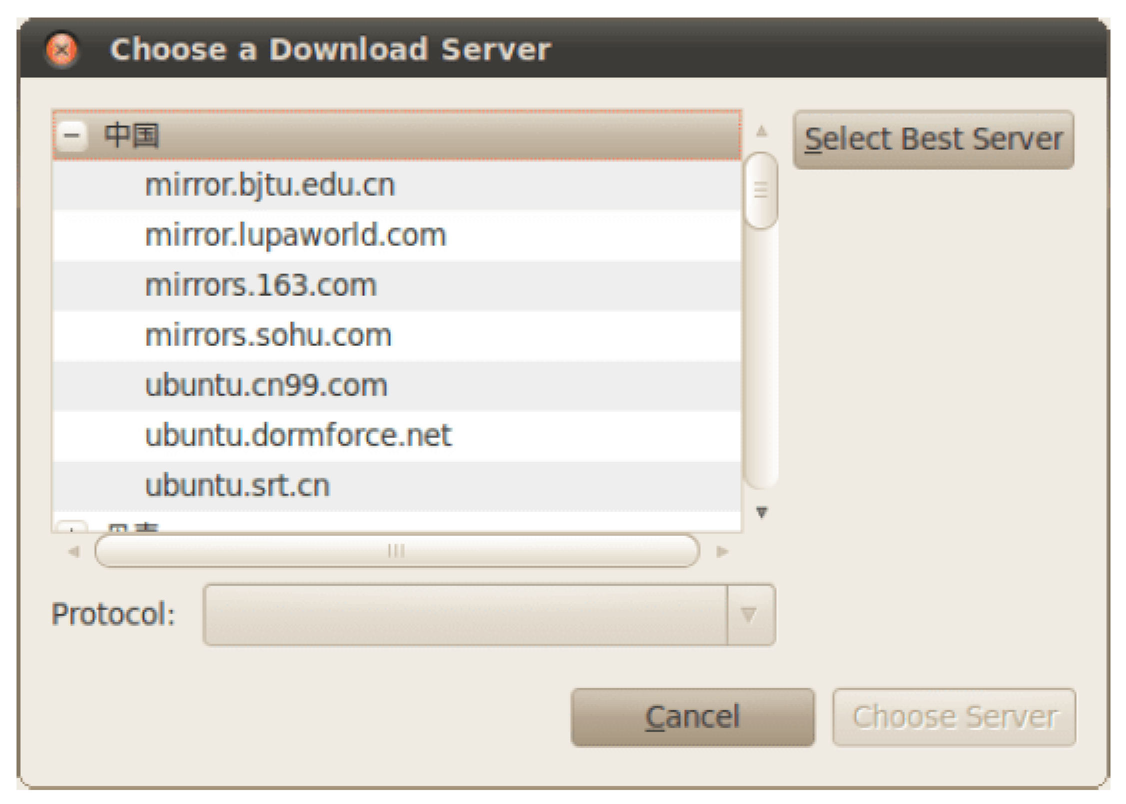


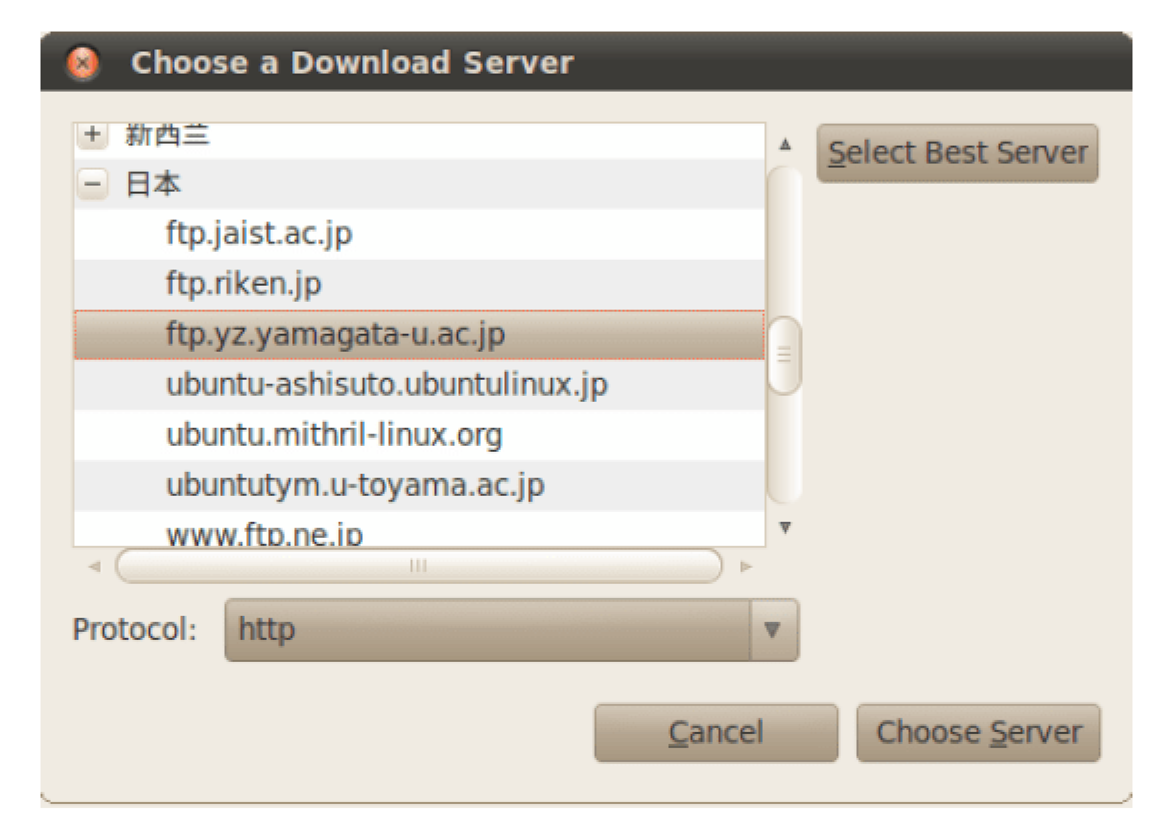
图 3.4　选择其他软件源

（4）选择之后会弹出一个如图 3.5 所示的窗口，让用户选择要用的软件源。用户可以直接在左侧的列表框里面选择，不过估计您会无所适从，不知道该选哪个。

（5）如果无所适从，单击右上角那个 Select Best Server 按钮，就会自动查找速度最快的软件源。

（6）查找结束之后，最快的软件源已经被选中，如图 3.6 所示，这时候只要单击 Choose Server 按钮就可以了。





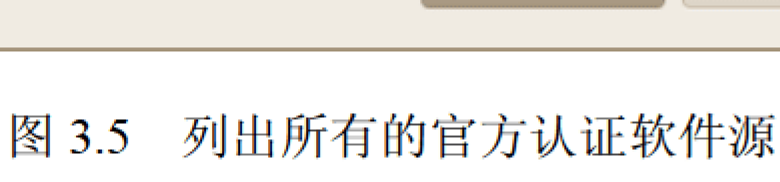

图 3.5　列出所有的官方认证软件源

图 3.6　找到的最快的软件源

提示： 最快的源不一定是地理位置上最近的，所以如果搜出个日本的源不要惊讶哦。

【方法 2，自己手动换】

我们这位懒蜗牛同学就没有用上面那种简便的方式，而是想尝试一下亲自动手。他先是通过 Firefox 上网找到软件源的地址（Firefox 是我们 Ubuntu 系统的默认浏览器，我喜欢叫她狐狸妹妹）。这并不难，用搜索引擎搜一下“Ubuntu 软件源”，很多地方都可以找到。

比如我们 Ubuntu 官方的 wiki 里就有：http://wiki.ubuntu.org.cn/Template:10.04source。软件源什么样？就是一大堆地址，类似这样：

```
deb http://mirrors.163.com/ubuntu/ lucid main universe restricted multiverse
deb-src http://mirrors.163.com/ubuntu/ lucid main universe restricted multiverse
deb http://mirrors.163.com/ubuntu/ lucid-security universe main multiverse restricted
deb-src http://mirrors.163.com/ubuntu/ lucid-security universe main multiverse restricted
deb http://mirrors.163.com/ubuntu/ lucid-updates universe main multiverse restricted
deb-src http://mirrors.163.com/ubuntu/ lucid-updates universe main multiverse restricted
deb http://mirrors.163.com/ubuntu/ lucid-proposed universe main multiverse restricted
deb-src http://mirrors.163.com/ubuntu/ lucid-proposed universe main multiverse restricted
deb http://mirrors.163.com/ubuntu/ lucid-backports universe main multiverse restricted
deb-src http://mirrors.163.com/ubuntu/ lucid-backports universe main multiverse restricted
```

找到之后要换源很简单，打开/etc/apt/sources.list 这个文件，把里面的东西清空，换上上面这一大坨就好了。软件源管理器做的其实也就是这样一个操作，新手还是用那个方便。像我这个懒蜗牛同学这样，非要自己动手改，结果，出问题了。他找到了一个看上去还不错的源，也知道要改哪个文件，于是，他动手了。他对命令似乎还挺熟悉，只见他运行了：

```
$gedit /etc/apt/sources.list
```

提示：输入命令的终端位于“应用程序”|“附件”|“终端”。

这个命令的意思是叫 gedit 起床干活，打开/etc/apt/sources.list 文件给用户看。于是我赶快去硬盘里叫醒 gedit 小弟。gedit 是一个文本编辑器，比 Windows XP 的记事本稍微强大点。gedit 接到命令后赶快打开那个文件，显示给用户。用户把里面的东西统统一删，把找到的软件源的地址粘贴了进去，然后猛然发现，怎么那个“保存”按钮是灰的呢？gedit 在那冷嘲热讽地念叨：“你以为你是谁呀？这可是重要的系统文件，你还想改？改坏了算谁的呀，你一个普通用户担待得起吗？”当然，这些都是他在工作间里自言自语，要真敢跟用户这么说话就等着被删吧。那么到底为什么不能保存呢？原因很简单——没有权限。

提示：/etc/apt/sources.list 文件里可以写入多个软件源，但 apt 在下载软件的时候不会同时使用多个源来加快下载速度，而是优先使用写在前面的软件源，只有该软件源里没有找到要安装的软件时，才使用下一个软件源，依此类推。

3.1.3　获取最高权限

“不对呀，你不是说安装系统的时候建的这个用户有超级用户的权力吗？”不好意思，您少记了 3 个字。我说的是，这个用户有“变身成”超级用户的权力。怎么变身？扭回头

来吧，不用看窗外的月亮，只需要在要执行的命令前面加上“sudo”命令就可以了。sudo 就是“以超级用户身份运行”的意思。用户直接运行 gedit /etc/apt/sources.list ，就是以普通用户身份打开 sources.list 文件，当然不能修改。应该运行：

```
$sudo gedit /etc/apt/sources.list
```

【变身技术指导】

这条命令的意思就是说：我要变身成超级用户并打开 sources.list 文件。但是不能您说变身我就让您变，您又不是变形金刚，这变身得讲条件。首先是核对一下身份，只有最初安装系统的时候创建的那个用户可以变身，装好之后再创建的其他用户就不行了（当然，最初的那个用户也可以把变身的能力赋予其他用户，这里说的只是默认情况）。确认了这个用户确实有变身的权利之后，还不算完，还需要让用户再输入一遍自己的密码。这样做，一来是防止恶意程序脚本骗取超级用户权限，再者也可以确认现在坐在电脑前的就是登录进来的用户。别回头老陈登录进来了，处理着半截照片上厕所去了。这时候来个修电脑的偷偷在老陈的电脑上以超级用户身份搞破坏，那就容易出事了。

提示：在命令行输入密码的时候没有任何回显，只要输入正确的密码并按回车键即可，千万不要怀疑自己的键盘过保修期了。

本以为我这位懒蜗牛同学会困惑好一会儿为什么不能保存，结果发现他好像不是不知道 sudo，只是一时忘记了，发现 gedit 不能保存之后，马上就把 gedit 关了，在命令前加上了 sudo 重新来了一遍，这回 OK 了。看来这家伙是个老手，只是一时忘了而已。我开始庆幸我能遇上这么一个用户了。

【变身练习，更新软件列表】

软件源修改了之后，还不能马上生效，得先通知超级牛力一声，让他去根据软件源更新软件列表。软件列表是什么意思？这个列表就是写明，现在所用的软件源里面都有什么软件，相互的依赖关系如何。这样当你要装软件的时候，超级牛力直接查看这个列表就知道相关的软件信息。否则，万一哪个财迷用户让超级牛力去装 give_me_money 软件，超级牛力还得跑到网络上找软件源服务器问：“您这有一个叫 give_me_money 的软件吗？”人家肯定没好脸色地说：“我还想要呢！没有，回去！”然后超级牛力再回来告诉用户，这样很耽误时间。所以，就要在每次换源之后，让超级牛力去获取软件信息，把这些信息存在硬盘上。以后用户要是再想装什么不靠谱的软件，就可以直接让他死了这条心了。那么怎么更新软件列表呢？很简单，还是需要变身的命令：

```
$sudo apt-get update
```

提示：成功运行一次 sudo 指令后，五分钟内再运行 sudo 指令不必输入密码。

好了，现在软件源也改了，列表也更新了。这回，懒蜗牛同学再打开那个 Language Support，提示安装的时候单击 Install 按钮——这回不用等到春节了，只需要十几分钟的时间就可以下载完了（具体时间长短还要看网络带宽）。

提示：以后每次对/etc/apt/sources.list 文件做了修改之后都要运行 sudo apt-get update 来更新软件列表。

另外，设置好软件源之后，再装别的软件就方便多了，想装什么软件只要运行：

```
$sudo apt-get install <软件包名>
```

就可以了，超级牛力会搞定剩下的事情。软件包名多数情况下就是软件的名称，不过也可能会有些出入，如果你想知道软件源里有没有你想要的软件，具体的软件包名称是什么，那么可以运行：

```
sudo apt-cache search <关键字>
```

这样就会列出所有相关的软件包了，这都是超级牛力更新了软件列表的结果。

提示： 除 sudo 命令外，还可以使用 su 命令来实现在命令行中提高权限。su 命令用于临时切换至任意用户——包括 root，但需要该用户的密码。例如运行：

```
$su user1
```

则可以临时切换至 user1 用户，并拥有 user1 用户的权限。但运行此命令后需要输入 user1 用户的密码。而 Ubuntu 系统默认的 root 密码未知，因此，需要配合 sudo 命令来运行 su，才可以切换至 root 权限。运行如下命令：

```
$sudo su root
```

如此，则相当于以 root 用户身份（sudo 提高权限的结果）运行“su root”，即要求切换至 root 自身。因此 su 命令不要求输入密码，直接切换至 root 用户。

3.1.4 为 apt 设置好网络

当然，刚才所说的这一切的前提是，你要把网络配置好。能上网，才能发挥超级牛力的能力。我所在的这台电脑因为是用那种家用的宽带路由器，什么 IP 地址、DNS、路由等，都是由路由器的 DHCP 服务自动分配的，所以在我这里不需要设置什么，把网线插上就能上网了。如果不是这样的怎么办呢？咱们分情况慢慢说。

【手动设置网络参数的有线连接】

有的地方没有 DHCP，需要自己手动设置网络参数，也就是 IP 地址、DNS 之类的。这个好办，你在 Windows 7 下怎么设置的来着？找网络连接是不是？那我这里还找这个就对了。

（1）单击界面上的 System（系统）|Perferences（首选项）|Network Connections（网络连接）打开网络连接界面，如图 3.7 所示。

（2）打开网络连接界面之后，选择 Wired（有线）标签（默认应该打开就是），下面的列表框里列出了所有的有线网卡，需要设置哪个，单击它选中，然后单击右面的 Edit（编辑）按钮就好了。

（3）点开之后就可以设置 MAC 地址，IPv4 的地址，甚至 IPv6 的地址。IPv6 用得还不广泛，咱就光说这 IPv4 吧。选择 IPv4 标签，出现图 3.8 所示的界面，在 Method（方法）下拉列表框里，选择 Manual（手动），就是手动配置 IP 参数的意思（原来可能是 Automatic

（自动），如果有 DHCP 就选这个）。

（4）选择手动后，下面的 Addresses（地址）列表框就有效了。单击右边的 Add（添加）按钮，然后在 Addresses 列表框里写上 IP 地址，子网掩码，网关。

（5）最后，再在下面的 DNS server 文本框里填上 DNS 的 IP 地址，就好了。什么？你问我具体应该填什么？跟 Windows 7 下一样！

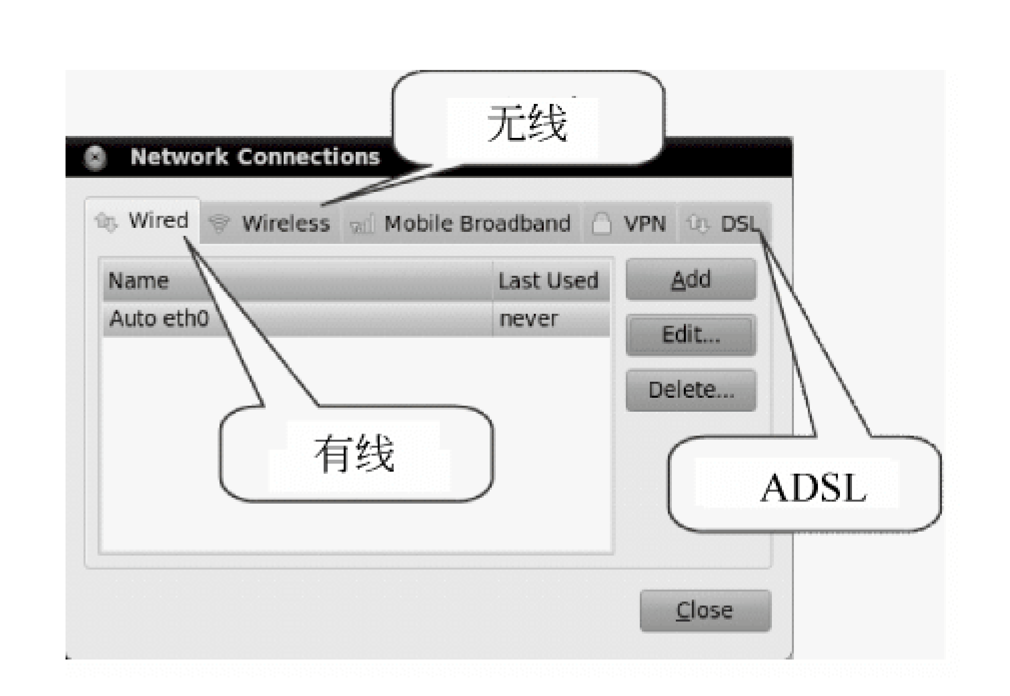


图 3.7　网络连接

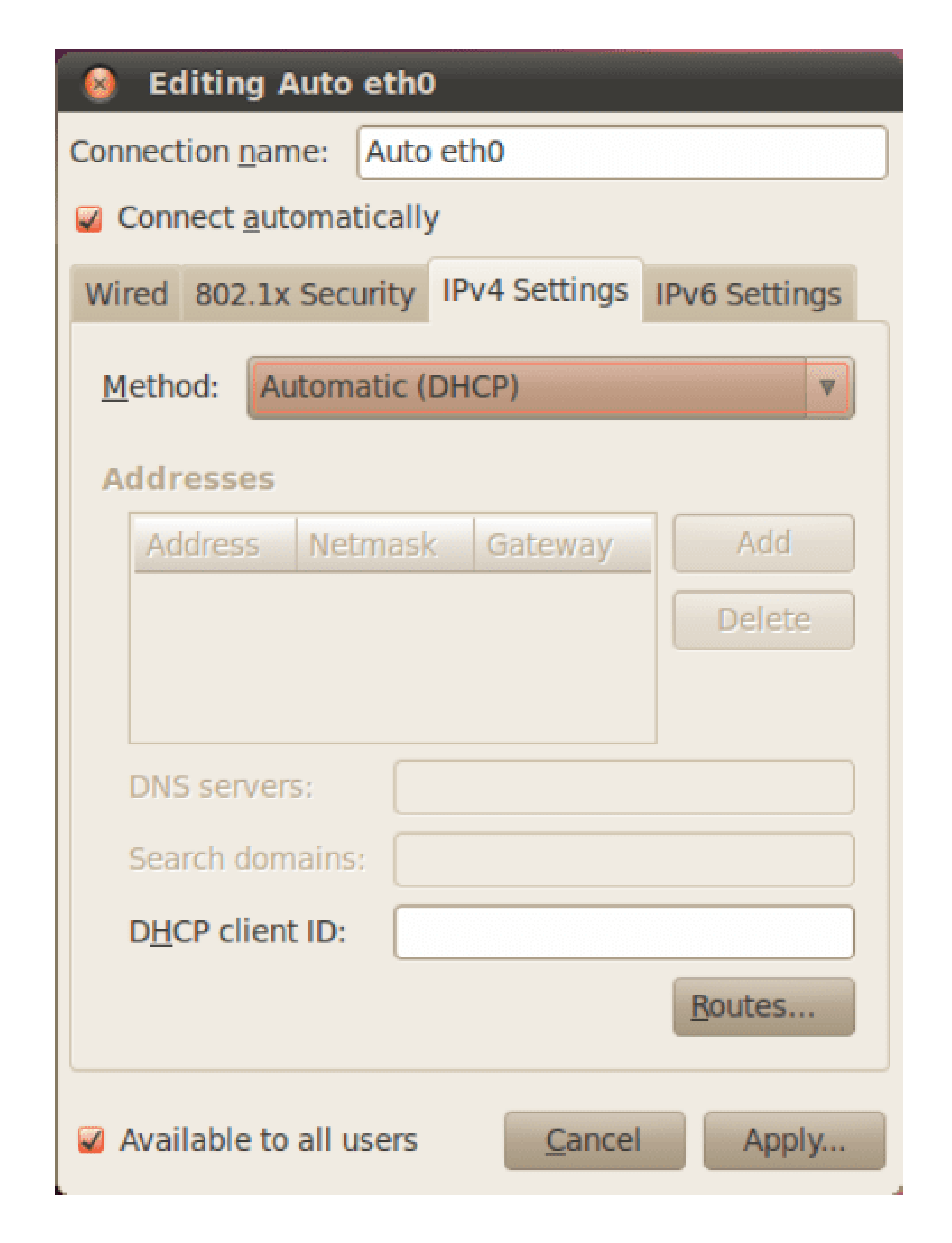


图 3.8　设置 IPv4 参数

【ADSL 拨号】

ADSL 这种拨号上网的方式是目前比较常见的，在我们 Ubuntu 系统里设置起来也比较方便。

（1）还是打开网络连接，切换到 DSL 标签。

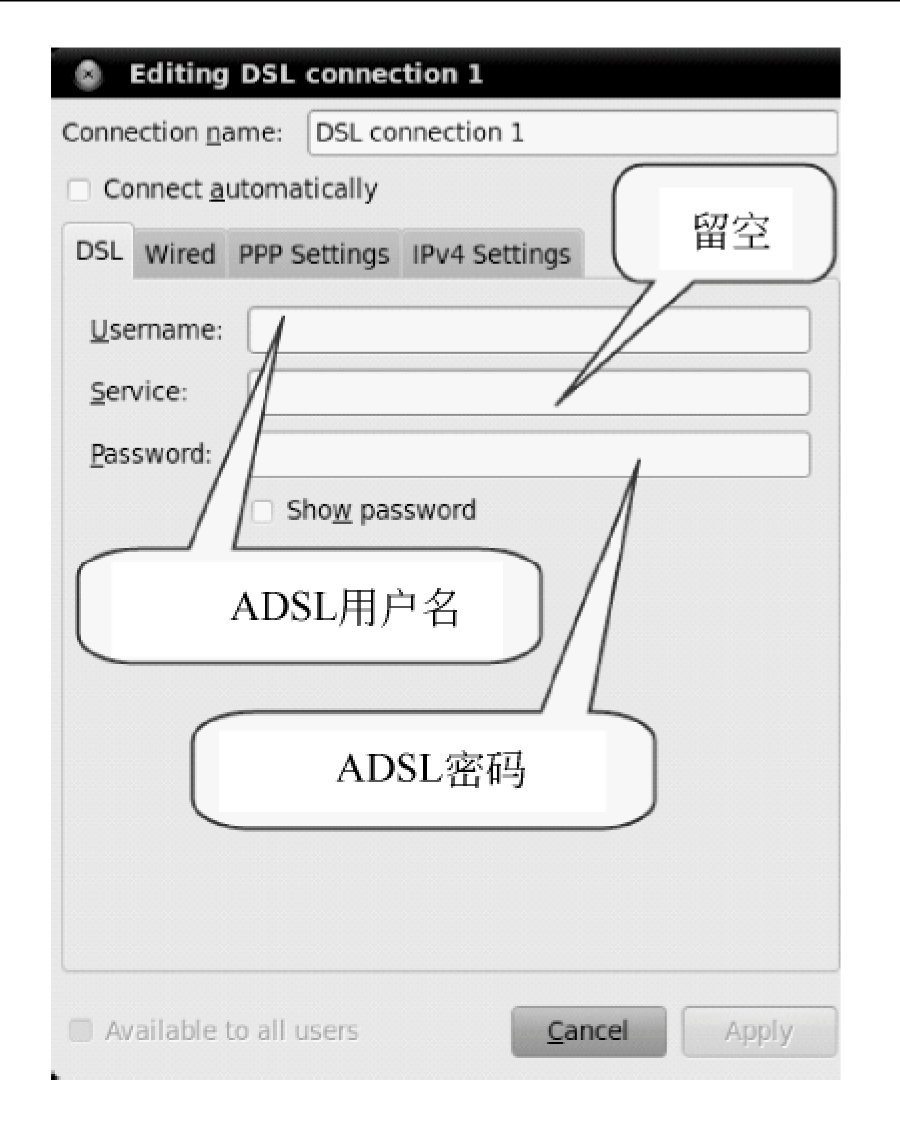
（2）单击 Add 按钮，出现图 3.9 所示的界面。

（3）在 Connection name 文本框里写入这个连接的名字。随便起一个就行，反正是给你们人类看的，我们操作系统不关心这个名字。

（4）如果希望在需要上网的时候自动连接到网络，就选中 Connect automatically（自动连接）复选框。

（5）在 Username（用户名）文本框中写上用户名，在 Password（密码）文本框中写上密码（当然是 ADSL 的用户名、密码，可别写你系统的用户名、密码），Service 文本框空着就好。都写好之后单击右下角的 Apply（应用）按钮。

（6）创建好以后，单击系统托盘上的网络连接图标（就是两个箭头那个），就可以在弹出的菜单里看到刚刚创建的 ADSL 连接了，如图 3.10 所示。单击菜单上的连接名，就可以连接到互联网了。



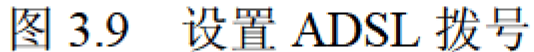

图 3.9　设置 ADSL 拨号



图 3.10　连接 ADSL

【WIFI 设置】

WIFI 现在也很普及了，如果电脑有无线网卡，直接单击系统托盘上的网络连接图标，就会显示出可连接的无线网络，选择一个单击一下就连上了，是不是很简单（当然，如果有密码还会让你输入密码）？

好，说了这么多，这会儿那个中文支持包也该安装完了，下一步还要设置点什么呢？

3.2　狐狸妹妹 Firefox

很快，中文支持装好了。懒蜗牛同学注销并重新登录了一下，总算看到了熟悉的文字。不过，这只是万里长征走完了第一步，还有很多的地方需要配置呢，比如浏览器就是其中之一。

3.2.1　安装 Flash 插件

从开始安装到配置好中文，已经过了很长时间，懒蜗牛同学喝口水，喘口气，忽然想起一件至关重要的事情——该收菜了！于是他再次叫醒狐狸妹妹 Firefox，向着他的菜地出发了。来到他的菜地放眼一望……咦？怎么什么也没有，一片空白啊？就算这菜被人偷光了，难道这地也让开发商征用了不成？再仔细抬眼一看，哦，原来狐狸妹妹已经做了解释，如图 3.11 所示。



图 3.11　Firefox 提示缺少插件

【让 Firefox 自动安装 Flash 插件】

问题其实很简单，就是没装 Flash 插件而已，那就装上吧。狐狸妹妹很贴心地在提示

缺少插件的同时，提供了一个“安装缺失插件(I)”按钮，就是图 3.11 右侧的那个。懒蜗牛同学单击了这个按钮，之后狐狸妹妹就跑到网上去找插件了。

狐狸妹妹安装插件跟超级牛力安装软件有点类似，都不需要您手动去下载，都是他们自己去网上找来装。但是狐狸妹妹的插件只在指定的官网上有，只此一家，别无分号，所以也不用专门为她设置什么软件源了。如果顺利，狐狸会在一阵搜索之后，找到适合的插件，并且自动开始安装，安装好之后，只要把狐狸妹妹重启一下，就可以看到效果了。

然而事情并不总是这么顺利的，比如这回，狐狸妹妹上网搜索了一遍之后向用户报告：“我搜了，可是呢……没找到，您自己看着办吧。”这件事情也得原谅狐狸妹妹，毕竟人家不是专业装软件的，所以装插件的时候难免有这样的时候。不过没关系，狐狸自动安装没成功，咱还有别的办法。

【用新立得安装 Flash 插件】

Flash 插件也是个软件，既然是软件，那就找超级牛力来装。我这位懒蜗牛同学也非常明白这个道理，于是他单击了“系统”|“系统管理”|“新立得软件包管理器”——等等，不是说要找超级牛力么，这个新立得又是个什么东西？

超级牛力确实很牛，干活没得说，但他是个命令行界面的软件，只能够通过文字跟用户交流。可是由于人类越来越懒，键盘能少敲一下就少敲一下，因此很多人并不喜欢跟字符界面的软件打交道，这时候，就该新立得出场了。新立得就是超级牛力的图形界面前端，用户通过喜欢的图形界面，把自己的意图告诉新立得，再由新立得转达给超级牛力。这两个家伙配合得很是默契，以至于很多人觉得，新立得就是超级牛力。

用户运行了新立得，由于新立得是用来给系统安装软件的，需要变身成超级用户才可以操作，因此新立得马上要求用户输入他的密码，核对身份，核对通过之后，才显示出新立得的界面，如图 3.12 所示。



图 3.12　新立得界面

在新立得的界面上装软件很简单，就像逛超市一样。

左侧上方的列表框里列出了所有的软件分类，左侧下方的几个按钮是分类方式。选中一个分类后，右侧上方列表框中就会列出这个分类里的所有软件包。选中其中一个软件包，下面就会有这个软件包的介绍（不过是英文的）。不过更多的时候，用户是使用上方的“快速搜索”文本框来查找需要的软件并安装的。

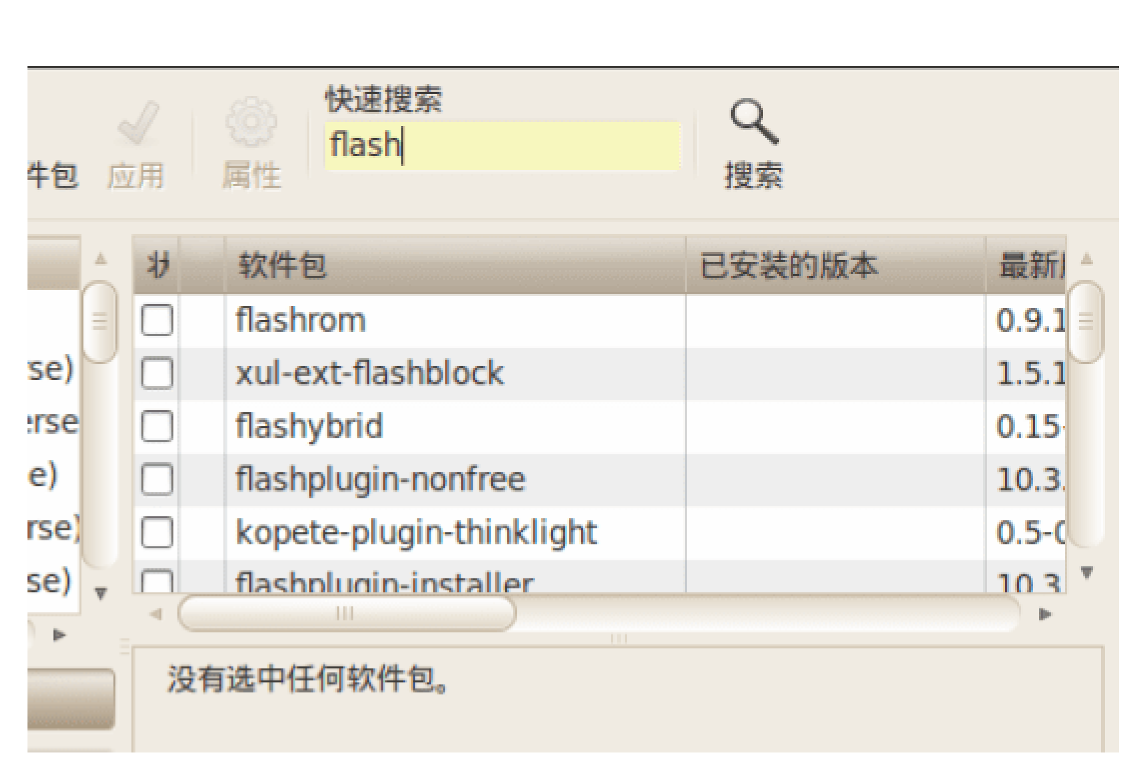
提示：软件包分类后面带有 universe 的，表示社区维护的开源软件；后面带有 restricted 的是官方维护的非开源软件；带有 multiverse 的是非官方维护的非开源软件；其余的是官方维护的开源软件。

装软件的时候还会有一些依赖关系。所谓依赖关系，就是要装软件 A，必须先装软件 B；就像要想用牙膏，就必须先得有牙刷（否则，总不能用鞋刷子刷牙吧）。

好，我们的懒蜗牛同学要装 Flash 插件了，我们顺便按照他的操作步骤来学习一下怎么用新立得安装软件。

（1）懒蜗牛同学先是在软件包分类列表框里选择了“全部”，然后在上面的“快速搜索”文本框里面输入“flash”，这就是说在全部软件包里查找关键词“flash”，于是很多跟 Flash 有关的软件包就列出来了，这时候他看到的就是图 3.13 所示的样子。

（2）懒蜗牛同学看到其中有一项“flashplugin-nonfree”，单击了一下，下边出现了这个软件包的介绍。他看了看，觉得没错，就是图 3.14 所示这个介绍。



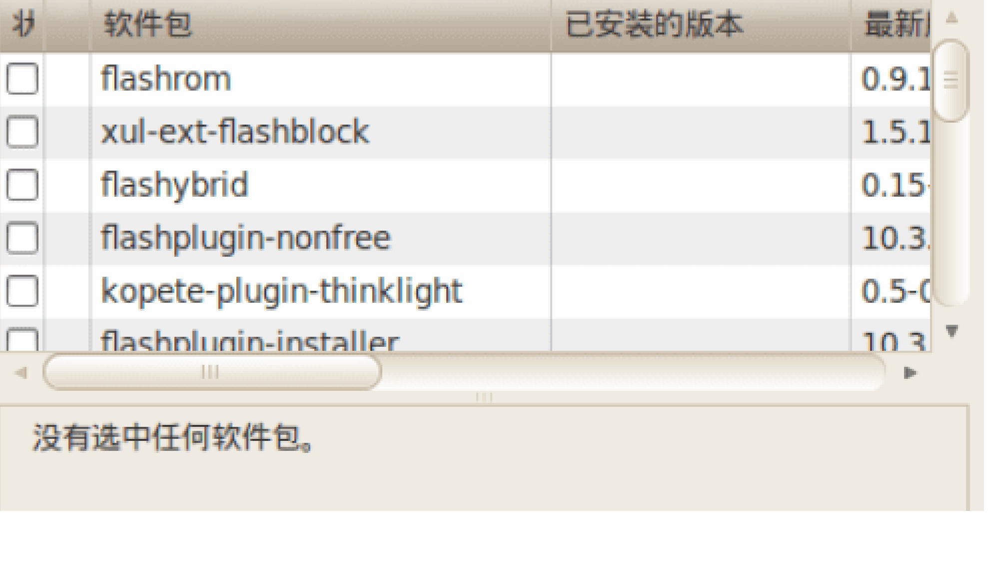


图 3.13　快速搜索软件包

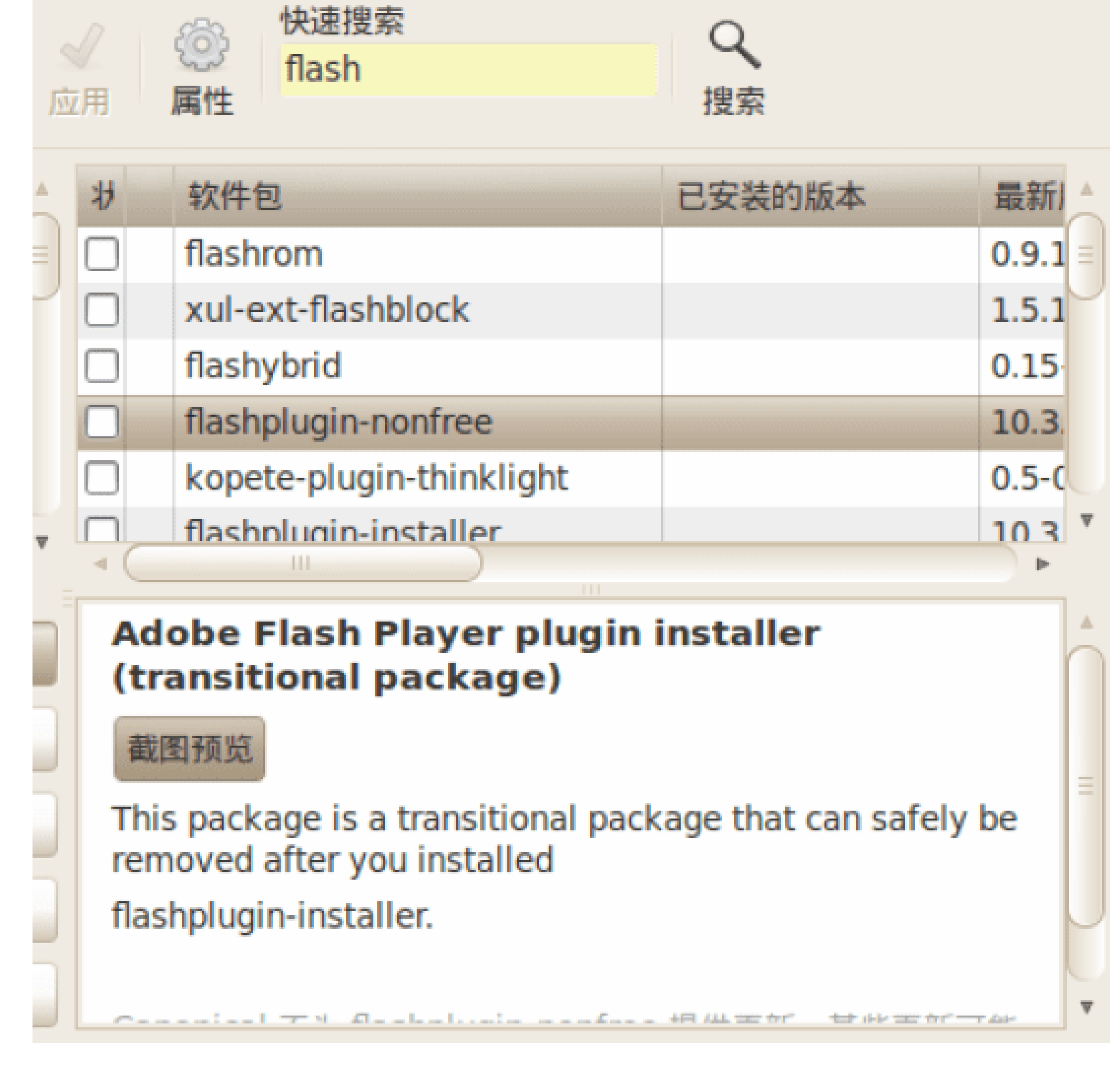


图 3.14　软件包详细信息

（3）他右击这个软件包，在弹出的快捷菜单里选择了“标记以便安装”。

（4）新立得收到命令之后，赶紧叫起超级牛力说：“牛哥牛哥，快起床，用户要装 flashplugin-nonfree 这个包啦。”超级牛力查阅了一下自己记录的资料，告诉新立得：“转告他，要装 flashplugin-nonfree，就得同时装上 flashplugin-installer、ia32-libs、lib32asound2、lib32bz2-1.0、lib32gcc1、lib32ncurses5、lib32stdc++6、lib32v4l-0、lib32z1、libc6-i386、nspluginwrapper 这些软件包。”新立得自然如实转告，就是图 3.15 所示这个界面。

（5）这位懒蜗牛同学倒是没有被这么多乱七八糟的包名吓倒，很淡定地单击了“标记”按钮。然后就开始安装了吗？没有，都说了新立得装软件就跟推着推车在超市购物一样，现在不过是把要买的东西放进购物车了而已。

（6）懒蜗牛同学看到了图 3.16 所示的这样的状况，该装的软件打好了标记，于是就单击了新立得界面上的“应用”按钮，这个按钮的意思就相当于结账。



图 3.15　依赖关系确认



图 3.16　单击应用开始安装

提示： 由于软件源里其实没有 64 位的 Flash 插件，因此 64 位系统在安装 Flash 插件的时候会安装 32 位的 Flash 插件和 nspluginwrapper，以及一些 32 位的库文件。原本 64 位的 Firefox 是无法使用 32 位插件的，但依靠 nspluginwrapper 可以使用 32 位的 Flash 插件。

经过一段时间的等待，超级牛力装好了 Flash 插件，并由新立得汇报给了用户。懒蜗牛同学迫不及待地牵着狐狸妹妹再次奔向了他的菜园。

【手动安装 Flash 插件】

软件源里的 Flash 插件只有 32 位的，即使您装的是 64 位系统，也只有 32 位的 Flash 插件可用。那么有没有办法安装上 64 位的 Flash 插件呢？有，那得手动安装，过程也不复杂。

（1）首先到这个地方下载与你的系统对应的插件：

```
http://labs.adobe.com/downloads/flashplayer11.html
```

对于我这个 64 位的 Linux 系统来说，就是选择 Download plug-in for Linux 64-bit。

（2）下载下来之后，是一个.tar.gz 的文件，直接双击解压出来，其实里面有用的就一个文件——libflashplayer.so。

（3）把这个文件放到/usr/lib64/mozilla/plugins 目录里，然后重启浏览器就可以了。

说起来容易，不过这个地方可不是随便一个用户就可以放文件的，得需要超级用户的权限才行。熟悉命令的同学可以在命令行里用：sudo cp 命令把文件复制过去，如果不会用 cp 命令，也没问题。按下 Alt+F2 键，在弹出的窗口中输入 gksu nautilus，如图 3.17 所示。

按 Alt+F2 键弹出的这个窗口是运行程序的，想运行哪个程序，在文本框里面敲就行了。比如可以在里面敲 firefox，然后单击“运行”按钮，就运行起狐狸妹妹来了。

🔔提示：Linux 系统中对大小写敏感，输入命令要确保大小写正确。如：应输入 firefox 而不是 Firefox 来启动火狐浏览器。

我们要运行的是 gksu nautilus，这个 gksu 和 sudo 具有一样的功能，让用户完成变身，只不过这是个图形界面的。nautilus 就是我们这里默认的文件浏览器。这段命令的意思就是用 root 身份打开文件浏览器。单击“运行”按钮之后，会让你输入密码，就跟打开新立得的时候一样。然后就可以看到文件管理器了，注意左上角这时候是图 3.18 所示这样了。




图 3.17　Alt+F2 运行程序



图 3.18　root 身份打开的文件浏览器

这个文件浏览器的窗口就拥有 root 的权限了，在这里面找到那个刚刚解压出来的 libflashplayer.so 文件，复制到/usr/lib64/mozilla/plugins 目录就可以了。复制完了记得关掉这个窗口，以免误操作。

🔔提示：64 位的 Flash 插件目前还在试验阶段，没有正式发布，因此可能会有各种未知问题。这也是软件源里为何不提供 64 位 Flash 插件的原因。

3.2.2　设置中文字体

【Flash 中的中文字体】

这回懒蜗牛同学来到菜地，一眼望去，果然看到绿油油的一片，庄稼长势喜人啊。咦？菜地里怎么这么多麻将牌呢？再一想，不对，哪有一地麻将牌的道理，仔细一看，哦，原来是所有中文都变成方块了。

懒蜗牛同学心里纳闷：我不是都装了中文了么，系统里的其他地方都变成中文了，怎么这里还是方块呢？百思不得其解的他，本着内事不明问老婆，外事不明问 Google 的基本方针，赶快让狐狸妹妹去 Google 了一下。原来，是 Flash 默认使用的字体不对，只要修改配置文件，换个字体就好了。

换字体不难，只要打开/etc/fonts/conf.d/49-sansserif.conf 文件，当然要记得加 sudo，因为是/etc/下的文件嘛，肯定只有 root 才有权限。然后把里面写明使用 sans-serif 和 serif 字体的地方，都换成 wqy-zenhei 字体，最后保存，就好了。改完了就类似下面这样（粗体为改动部分）：

```
<?xml version="1.0"?>
<!DOCTYPE fontconfig SYSTEM "fonts.dtd">
<fontconfig>
<!--
  If the font still has no generic name, add sans-serif
 -->
    <match target="pattern">
        <test qual="all"

name="family" compare="not_eq">
            <string>wqy-zenhei</string>
        </test>
        <test qual="all" name="family" compare="not_eq">
            <string>wqy-zenhei</string>


    </test>
        <test qual="all" name="family" compare="not_eq">
            <string>monospace</string>
        </test>
        <edit name="family" mode="append last">


<string>wqy-zenhei</string>
        </edit>
    </match>
</fontconfig>
```

那么换上的这个 wqy-zenhei 是个什么字体呢？这就是在国内开源界大名鼎鼎的文泉驿正黑字体，这是一个可以免费使用而且效果还不错的字体。

【网页中的中文字体】

除了 Flash，狐狸妹妹显示的网页中的字体也可以设置。这就更简单了，运行 Firefox 之后，选择“编辑”|“首选项”，在弹出的窗口里选择“内容”标签，如图 3.19 所示。



图 3.19　修改 Firefox 网页字体

在这个页面中间的“默认字体”下拉列表框里选择你想用的中文字体就可以了。不过也没多少可选，主要就是文泉驿那几种字体还不错。有人会觉得这样太单调了，就这么点字体可选。别着急，字体不多，咱可以装嘛。

【安装更多的中文字体】

字体也是软件的一部分，按说也可以用超级牛力来装。不过，软件源毕竟不是中国人开的，里面基本没有更好的中文字体了，怎么办？

如果您装了 Windows 7 或者其他商业化的操作系统，他们肯定是带有不少中文字体的。可以直接把他们的字体拿过来用（前提是你的系统是正版的，你已经为那些字体付过费了，否则就是偷来用了），步骤也简单。

（1）Windows 7 的字体一般放在 C:\Windows\Fonts 这个文件夹里，里面是一些.ttf 或者.ttc 的字体文件，把你想要的字体对应的文件复制到我们 Ubuntu 系统里面来，随便存在哪都可以。

（2）进入我们 Ubuntu 系统，双击刚刚复制过来的字体文件，就会出现如图 3.20 所示的窗口（话说我很怀疑设计这段程序的人练过气功）。



图 3.20　字体预览和安装

（3）左侧显示的是这个字体的效果（这里我们就用文泉驿字体做示例了），右侧是一些基本信息。确认这是你需要的字体后，单击右下角的“安装字体”按钮，字体就安装好了，简单吧。

总之，安装字体，就是先得到.ttf 或者.ttc 文件（从别的系统复制也好，网上下载的也行），再双击字体文件，单击“安装字体”按钮，就可以了。

3.2.3　扩展阅读：文泉驿的诞生

看过了上面的故事，有的同学可能会发牢骚说：你们这个 Ubuntu 系统，怎么就不能多带点中文字体呢？怎么就只有那么个文泉驿？这个文泉驿又是什么来路？

好，各位要有兴趣，我说说，您听听，在想当初……

【忆往昔艰苦岁月】

很久很久以前（对你们人类来说其实也不算久啦，也就六七年以前），那时候的 Linux

日趋完善，不少国内的开源同行们，都来尝试安装各种各样的 Linux。虽然硬件兼容的越来越多，应用程序的安装越来越便利，但炎黄子孙们安装了 Linux 之后无一例外地遇到了中文化的问题——没有一个合适的中文字体。

要知道，Linux 是自由的，开源的，其中很多是免费的。那么自然不可能在免费的 Linux 中带有任何需要付费的字体。那么 Linux 上自带的免费字体是从哪来的呢？都是世界各地热心的爱好者们贡献的。爱好者们自己创作一套字体，然后无偿贡献出来，给大家使用。因此，有很多优秀的、免费的英文字体、法文字体、德文字体等。

可为什么其他国家有热心爱好者贡献字体，就没有热心的中国人贡献字体呢？是因为中国人懒吗？不是；是因为中国人自私？也不是；是因为中国人口少？你自己信么？答案很简单，因为中国字多！人家做一套英文字体，总共也就 26 个字母的大写加小写，外带 10 个数字和一些标点符号，加在一起不超过 100 个。一个人花一周时间就能做完了。汉字有多少？找本新华字典翻开前言看看——收录汉字一万个左右！而且除了“一”、“丁”、“乙”这样极个别的特例以外，是个汉字就比英文字母难写吧。要是让一个人把这一万个汉字都做成电脑中的字体，还不得累吐了血啊。就算是只做常用汉字也得有 4000 多个，这还不算繁体字和各种数字、标点。

那么那时候有没有中文字体呢？当然是有的，否则难道十年以前中国人都不用电脑？可是一般中文字体都是需要收费使用的——这个很合理吧，这么困难的东西，人家凑几个人费了挺大劲做出来了，人家也得穿衣吃饭，养家糊口嘛。就算有几个免费的中文字体，也有很多问题：丢字啊、难看啊之类的。所以那个时候，Linux 的中文用户就只有忍受着质量差，总丢字的中文字体，或者把其他系统下的付费字体复制过来用。

【文泉驿横空出世】

直到 2004 年，中文字体的事情有了转机，带来转机的，是一位美国哈佛大学医学院的博士。

这位仁兄倒不是来发扬国际主义精神的，他之所以关注中文字体，是因为他本身是个中国人——房骞骞博士。不知道是不是因为他家房子被拆迁了，所以就去美国当博士去了（这都挨得着么……）。反正他在接触到 Linux 的时候，发现缺少中文字体是件很头疼的事情。他也知道要想自己做出一整套中文字体，那是一定要累吐血的。但是他还知道，积少成多，集腋成裘，积跬步成千里，积点水成江河……（此处略去类似成语、俗语若干）

他利用网络为平台，创建了“文泉驿”项目，目的是要创作出一套高质量的、免费的中文字体。他自己动手编码，设计了一个网站，简化设计字体的复杂程度，把汉字字体的绘制工作搬到了网页上。这让每一个热心的志愿者都可以按照网站上的指导，完成一个个汉字字体的绘制。就这样借助全球草根志愿者的力量，他开始了“万里长征”。经过数年的连续奋战，终于取得一项永留史册的硕果，这就是“文泉驿”汉字库，其中包括点阵和矢量字体。我们系统里的文泉驿正黑，就是其中之一。

目前，文泉驿项目依然在不断地完善，不断地创造新的字体。如果您有兴趣，可以去 http://wenq.org/画几个汉字，为开源的字体贡献一点点力量。画起来也不难，网页上的操作界面大约是图 3.21 所示的这个样子，稍微熟悉一下就可以上手了。

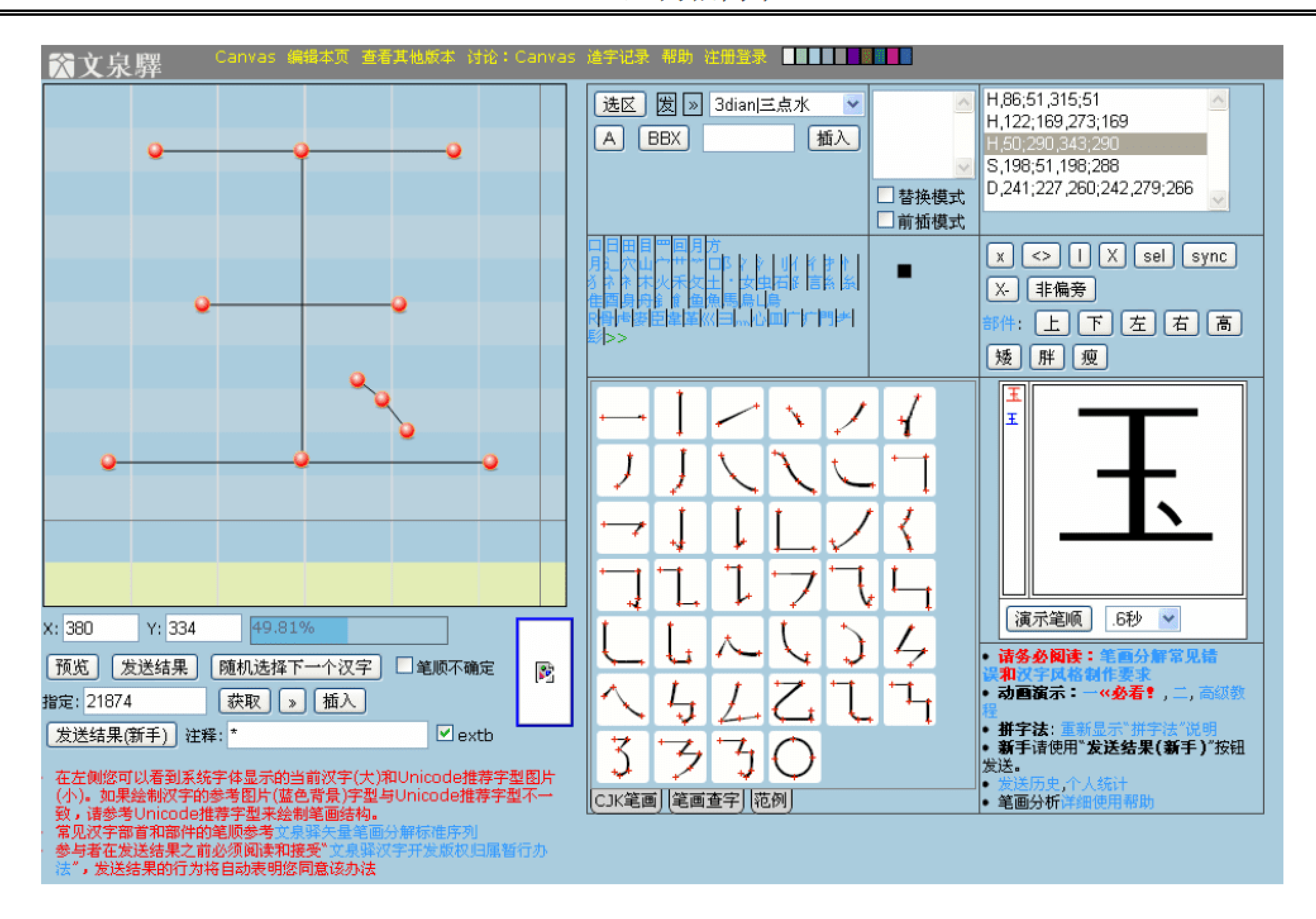


图 3.21　文泉驿字体设计界面

3.3　心有灵犀 Empathy

弄好了 Flash 和字体后，懒蜗牛同学很高兴，他想把这个好消息告诉给他的好友，忽然意识到一个问题——聊天软件在哪里呢？聊天软件这么重要的东西我们肯定是要有的，就在“应用程序”|“互联网”里面，有个“Empathy 即时通信客户端”，就是默认的聊天软件。

3.3.1　集 Gtalk、MSN、Icq 等于一身的 Empathy

Empathy 这个家伙其实就是个送信的，如果你查字典，会发现 Empathy 这个词的解释是：感情移入，同感（对他人的感情，经历等的想象力和感受力）。解释得比较复杂，于是我们就叫他“心有灵犀”吧（以下“心有灵犀”特指 Empathy）。

心有灵犀这家伙虽然送信，但他并不负责送那种长篇大论的 E-mail，而是负责发短消息——short message，也有叫短信的。不过别误会，这可不是指手机的短信，而是指像 MSN、Gtalk 这类的即时通信软件的消息。这些聊天软件的工作都是送信，使用者把要发送的消息敲出来，像一封信一样交给他们，他们把信放在信封里——这个过程叫打包，然后把这个包发送给对方的软件。对方软件拿来这个包，先要拆包——也就是把信从信封里拿出来，然后把里面的内容显示给用户看。

可是这些软件互相之间是不能通信的，MSN 不能给 Gtalk 发消息，反过来也不可能。这个现象估计不用我多解释。原因主要是因为他们的通信协议——也就相当于信封的打包

方式不一样。比如 MSN 的信封可能是从上面拆开取信；Icq 的信封则是从侧面拆开取信；Gtalk 的信封可能是用订书器订上的，需要拆钉取信；而雅虎通的是用胶水粘上的，需要涂水溶胶取信，反正各有各的高招。那么心有灵犀呢？他全会！否则都对不起他的名字。

心有灵犀在学校的时候，专门学习过各种各样的通讯工具信封的封拆方法。所以，心有灵犀可以同时支持很多种聊天软件，不用再同时开着 Gtalk、还开着 MSN、再开着 Icq 了，只要开一个 Empathy 就都能聊了。

现在，用户已经单击了“应用程序”|“互联网”|“Empathy 即时通信客户端”，我得赶快去叫醒心有灵犀起床干活了。

3.3.2　Empathy 的账户设置

心有灵犀来到了工作间，这是他第一次运行，所以先像图 3.22 这样很有礼貌地跟用户打了个招呼，介绍了一下，说明自己是用来聊天的软件，可以支持好多种协议，并且问用户：您有相应的账号了吗？

懒蜗牛同学说：有。当然，他不是拿嘴说的，而是用鼠标单击了“有，我现在就输入我的账户信息”这个单选按钮，然后单击“前进”按钮。

然后，心有灵犀就又问了：那您打算用什么账号聊呢？并且给出了一个下拉列表框，里面是他可以支持的所有协议，让用户选择，如图 3.23 所示。

提示： 虽然可以看到 Empathy 给出的账户类型中包括 QQ，但由于 QQ 协议经常更新，因此，Empathy 所带的 QQ 协议已基本不能使用。



图 3.22　Empathy 首次运行



图 3.23　账户类型选择

懒蜗牛同学选择了 MSN，选择之后，下方的内容就变成了 MSN 相关的设置，其实也没什么，就是账户和密码。懒蜗牛同学在下面填写好了自己 MSN 的账户和密码，发现心有灵犀在窗口下面还有问题在问：您还要设置其他聊天账户吗？懒蜗牛同学想了想，嗯……待会儿还需要设置 Gtalk 的账号。于是就单击了“是”那个单选按钮。设置好了就是图 3.24 这样。

设置好了之后单击“前进”按钮，然后就出现了和图 3.23 同样的一页，这回懒蜗牛同

学如法炮制，设置好了 Gtalk 的账号，然后单击了下面那个“没有，就这样吧。”单选按钮，最后单击“前进”按钮，心有灵犀的界面就变成图 3.25 所示这样。

心有灵犀继续问：您看，我还可以进行局域网通信，要是您的内网里也有用同样软件的，就可以聊天了，就跟飞鸽似的，要不要开启这个功能呢？懒蜗牛同学心说你怎么这么烦人呢？跟推销的似的，我就这么一台电脑，根本用不着这个功能啊，就勾上了那个“暂不启用此特性”的复选框，一口回绝了。心有灵犀也是见好就收，再没废话了，赶快干正事。不一会儿，懒蜗牛同学就看到了久违的 MSN 和 Gtalk 上的好友，尤其是那个很聊得来的 MM，呵呵，赶快去打个招呼吧。这一打招呼，坏了……



图 3.24　MSN 账户设置

图 3.25　局域网聊天设置

3.3.3　配置输入法

【默认的 IBus 输入法平台】

“这是什么破输入法啊？怎么都打不出字呢？”对于这一点，我比较惭愧，说来还是我中文课没学好，写得不好不说（显示乱码），认起来也有问题（输入法不好用）。

话说我们学校自从上一届，也就是 Ubuntu 9.10 开始，将 IBus 作为默认使用的输入法平台。所谓平台，就是说这个 IBus 只是个输入法的框架，上面可以安装各种输入法，装上五笔就是五笔，装上拼音就是拼音，装上日文输入法就可以打日文了。

IBus 是个比较新的输入法平台，所以有很多不成熟的地方，有许多热心的人为他开发输入法，光拼音就好多种。默认的这个不太好用，换一个就好了，换起来也很简单，咱们一步一步来。

（1）单击“系统”|“首选项”|“IBus 首选项”菜单，就弹出了如图 3.26 所示的设置 IBus 的窗口。

（2）选择“输入法”标签，会看到下面的“输入法”列表框里面已经有了很多输入法，光拼音输入法就不少，如图 3.27 所示。不过别看默认装了这么多，没一个好用的。我们还得再装个靠谱的拼音输入法。

提示：输入法列表框中排在第 1 位的为默认输入法，即按下 Ctrl+Space 组合键呼叫出来的输入法。

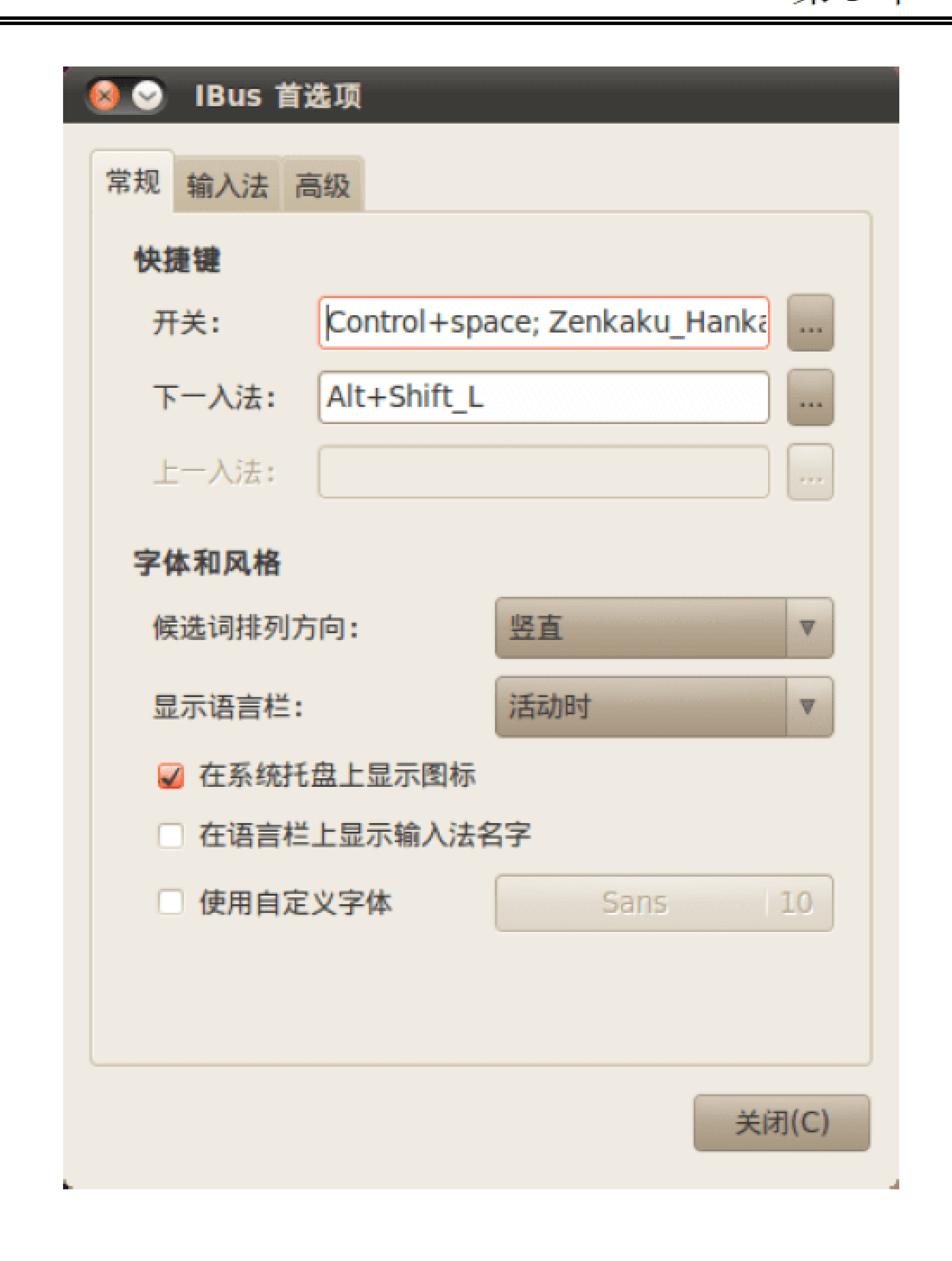


图 3.26　IBus 设置



图 3.27　输入法选择

（3）单击上方“选择输入法”下拉列表框边上的下三角按钮，会列出各种语言的级联菜单，我们自然是选择“汉语”。鼠标指向了汉语之后，就会弹出各种输入法的子菜单，这时候，一定要选择名字是“Pinyin”，图标是一个大“拼”字的那个输入法。就像图 3.28 中那个（怎么好像那些不好用的都给装上了，正经有用的都没装呢，谁设计的？）。

（4）选择之后，Pinyin 输入法就出现在了上方的下拉列表框的位置，如图 3.29 所示。这时候单击右侧的“添加”按钮，这个输入法就被加到下面的“输入法”列表框里了。





图 3.28　选择拼音输入法

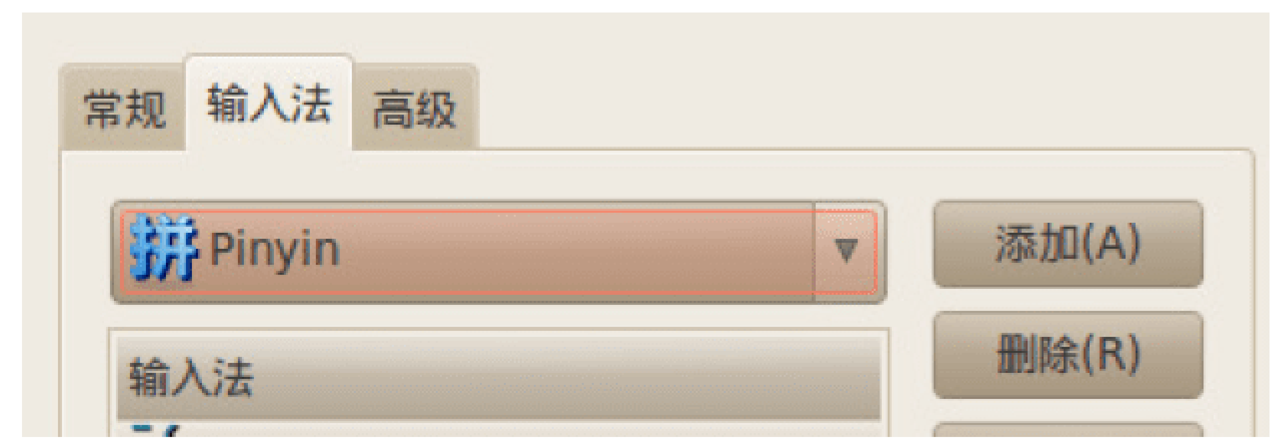


图 3.29　添加 Pinyin 输入法

（5）为了更清晰简洁，可以把其他不用的输入法都删除。具体方法就是单击选中“输入法”列表框里不需要的输入法，然后单击“删除”按钮，于是，如图 3.30 所示，世界清静了。



图 3.30　只保留需要的输入法

【老牌的 SCIM 输入法平台】

这个 IBus 毕竟混输入法这一行的日子还不长，所以有些缺心眼的行为还是可以原谅的。在 Ubuntu 9.10 之前的版本，都是用的 SCIM 输入法平台，这是个老牌的输入法平台了，问题相对少很多。很多老用户比较怀念这个输入法，没关系，在我们 Ubuntu 10.04 里也是可以用的，装上就行。

（1）要想用 SCIM，只要安装“scim”和“scim-chinese”这两个包就可以了。用图形界面的新立得也可以，用超级牛力也行。如果用超级牛力就是这个命令：

```
$sudo apt-get install scim scim-chinese
```

其中 scim 包是主程序，输入法的框架。scim-chinese 是中文输入法，其实就一个智能拼音。

（2）安装这两个包会有很多依赖关系，不用管它，全都安装就可以了。

（3）安装好之后，需要切换输入法平台，从 IBus 变成 SCIM，切换很简单，只要运行命令：

```
$im-switch -s scim
```

就可以了。im-switch 就是用来切换输入法框架的命令，用着不爽了想换回 IBus 就运行：

```
$im-switch -s ibus
```

很方便吧。不过运行 im-switch 后不会马上生效，需要注销当前用户并重新登录一下，就看到效果了。

提示：SCIM 有时候会和一些基于 Qt 库开发的程序发生冲突，导致无法正常使用。这也是为什么 Ubuntu 的默认输入法换成了 IBus 的原因。

如果你不喜欢用命令行也没关系，图形界面也能切换输入法。就在“系统”|“系统管理”|“语言支持”里面，打开就看到下方有一个“键盘输入方式系统”下拉列表框，在那里面选择你想要的输入法平台就可以了，如图 3.31 所示。

不过我们的懒蜗牛同学觉得 IBus 还是挺称职的，并没有换成其他输入法。这会儿正用 IBus 跟 MM 聊得热火朝天呢。



图 3.31　切换输入法框架

3.3.4　Linux 下的 QQ

输入法弄好了，我们的懒蜗牛同学使用心有灵犀跟朋友们聊得正欢，忽然想起一件事情，怎么聊 QQ 呢？毕竟懒蜗牛同学的很多朋友都在用这个软件，不能上 QQ 会失去和很多好友的联系。不过要想在我们 Ubuntu 系统里聊 QQ 还是有点难度的，因为这个家伙的协议不公开。

【热心爱好者的 QQ】

像 MSN、Gtalk 这些聊天软件，他们都使用公开的协议（至少基本的文字聊天部分是公开的，什么视频、语音之类的功能单说）。有专门的文件写着自己只收什么什么样的信封。比如必须蓝色，上面印着蝴蝶，上开口直接撕开的信封才能发给 MSN。因此，心有灵犀才能学习他们信封的打包方式，从而和这些聊天软件们互发消息。可是 QQ 这家伙的信封的封装方式却从来不告诉别人，而且很复杂，上面有乱七八糟的很多防伪标识，如激光、磁条、水印，比人民币还热闹。以前，Linux 下聊 QQ 主要有两个软件：Java 编写的 LumaQQ 和基于 Qt 库的 EVA。他们的作者通过黑盒研究，抓包测试，猜出了 QQ 信封的部分封装方式，因此可以和 QQ 通信。图 3.32 所示就是 EVA 的运行界面。但是 QQ 信封的封装方式不但不公开，还经常升级，搞得这两个软件也经常不能用。

提示：Qt 库是 KDE 桌面环境使用的图形开发库，而 Ubuntu 默认使用基于 GTK 库的 Gnome 桌面环境。在 Gnome 环境中运行基于 Qt 库的软件，启动速度会比较慢，但不影响功能和运行速度。

【官方原生的 QQ】

不过现在好了，因为有了官方的、原生的、正宗的，QQ for Linux，如图 3.33 所示。

QQ for Linux 和心有灵犀一样，也是一个即时通信软件，我就不多介绍了。不过可别想着让超级牛力去装什么 QQ，他肯定告诉你：没有！因为 QQ 不在我们的软件源里，要

想装，得让狐狸妹妹直接去 QQ 的网站上下载，就在这里：

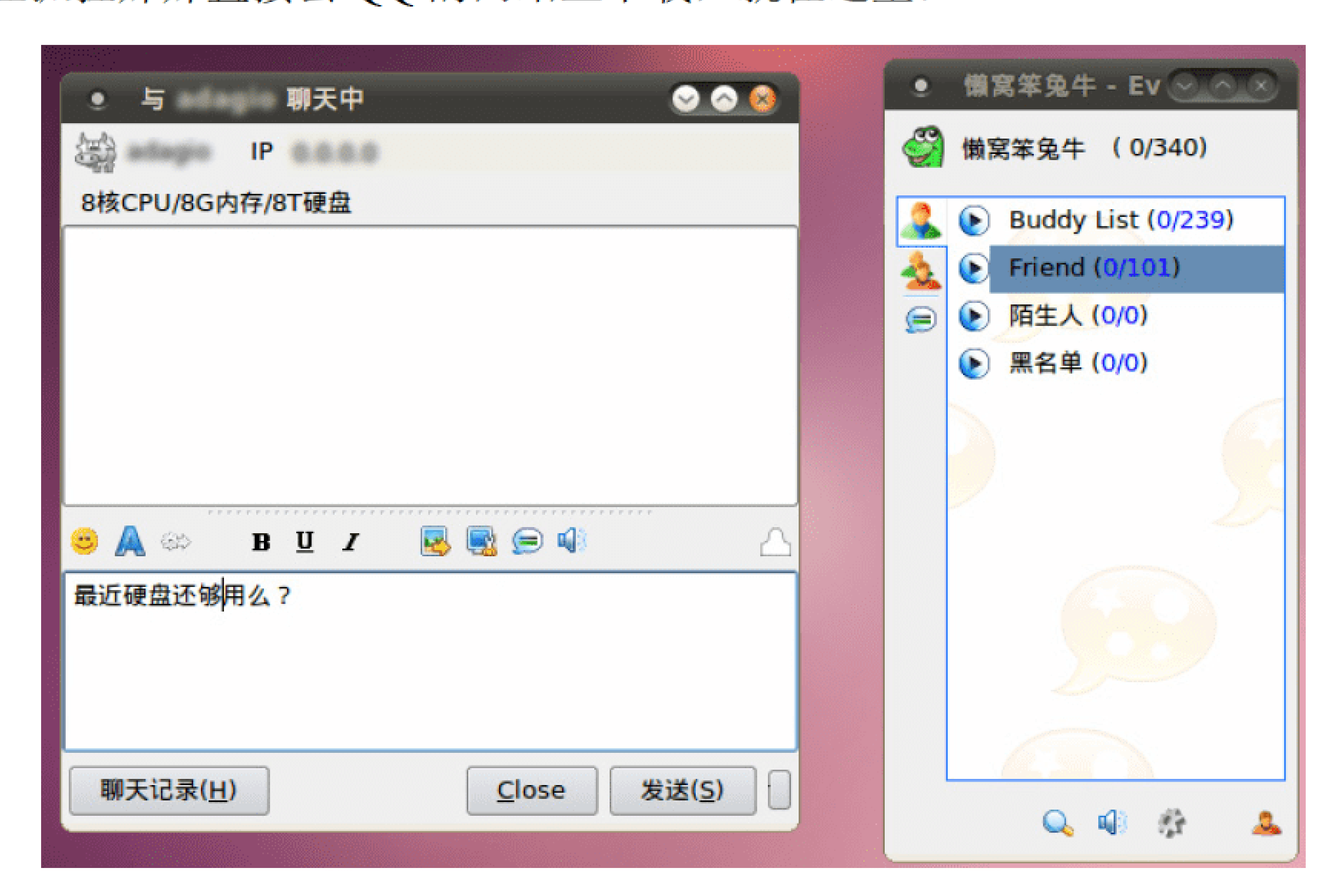


图 3.32　EVA 运行界面

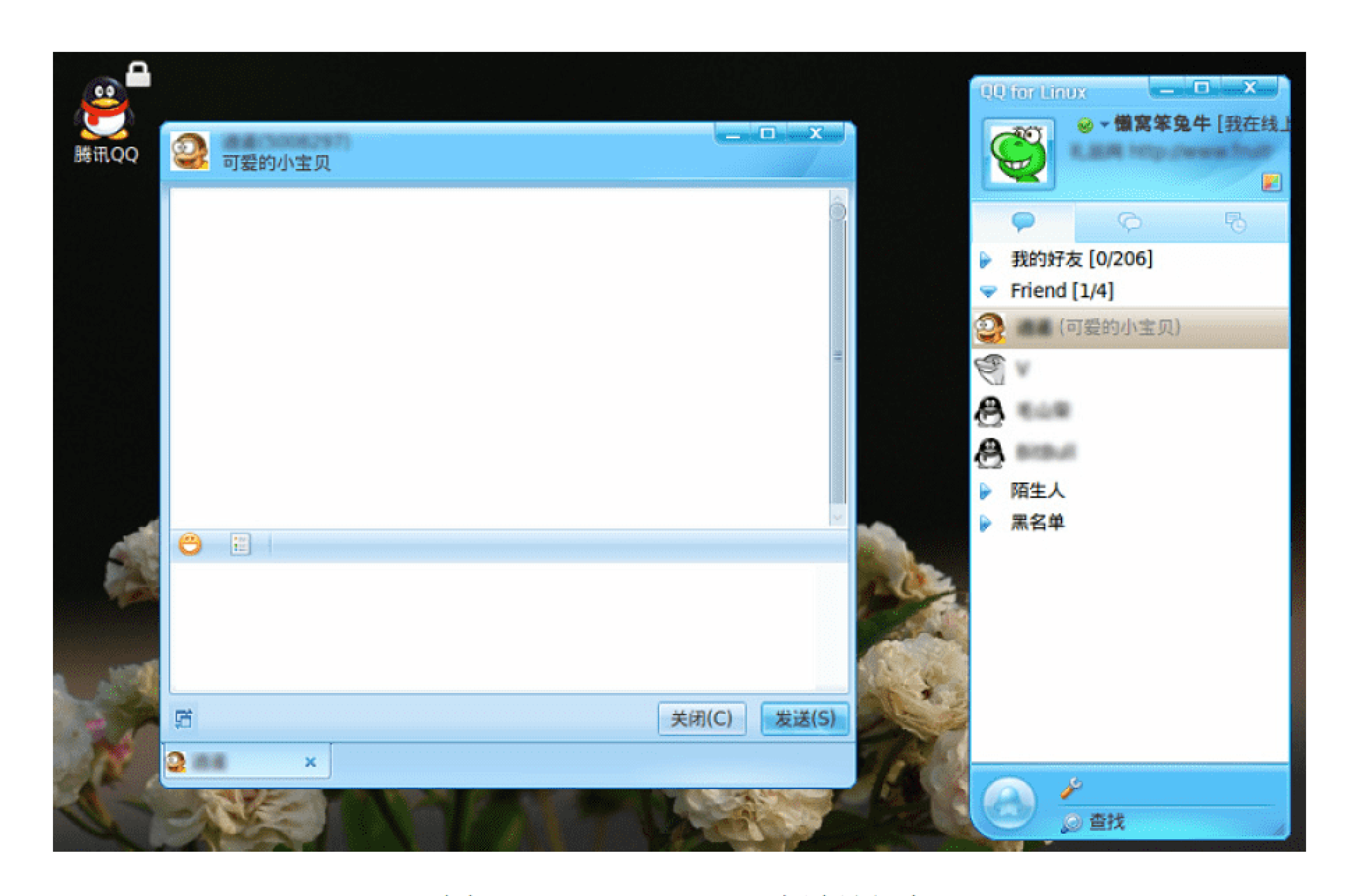


图 3.33　QQ for Linux 运行界面

```
http://im.qq.com/qq/linux/download.shtml
```

记住一定要下载最左边那个 DEB 包的，那个是我们 Ubuntu 系统可以支持的，并且安装最为简单，双击就好。

这 QQ 是一个叫做腾讯的公司做的。话说这个腾讯啊，看人家 Icq 软件玩得挺火，于是也弄了个 Oicq，抢占了国内市场，结果一发不可收拾，毕竟那时候国内的网络还刚刚起步，这东西新鲜啊，于是全民 Oicq，大家上网聊天。后来 Oicq 改名，叫 QQ 了，可是一直以来，由于各种原因，腾讯这个公司只能做出 Windows 版本的 QQ 来，Linux 下的没有。

现在腾讯公司不知怎么想开了，提供了 Linux 版的 QQ，虽然功能也很简陋，不过，文字聊天，传个图片，收个自定义表情什么的还是没问题的，总算是个好的开始吧。

提示：目前 QQ for Linux 维护力度很弱，更新进度很慢，在 Ubuntu 11.04 以后的系统中使用时可能会出现问题。

【网页版的 QQ】

QQ for Linux 虽然能用，但是也有不少的毛病。比如经常莫名其妙地占用硬盘，只要他开着，硬盘就经常工作着，也不知道读什么呢，所以很多人心里没底。于是就用网页版的 QQ 代替了。网页版的 QQ 也是官方的，所以不用担心哪天用不了，而且使用也方便，不用装软件，只要用浏览器打开 web.qq.com 网页，登录进去就可以了，功能比 QQ for Linux 还多。如图 3.34 所示就是网页版 QQ 的界面。



图 3.34　网页版 QQ 界面

3.4　多　媒　体

聊天工具和输入法搞好了，用户终于和 MM 幸福地聊在了一起。俩人聊着聊着，聊到了音乐。最近 MM 在听一首歌，感觉很不错，就介绍给懒蜗牛同学。而他自然愿意主动贴近 MM 的生活，于是赶快上网下载这首歌曲来听。

看来，到了我们的多媒体部门出场的时候了。

3.4.1　安装解码器

狐狸妹妹很麻利地下载到了这首歌，是一个扩展名是.mp3 的文件。懒蜗牛同学很自然

地就双击了这个文件，想要播放来听听。播放音乐这种事，自然要找多媒体部门，“电影播放机”就是其中的一员，他的英文名字叫 Totem。

【播放音乐遇到的问题】

我看到用户要播放 MP3，就赶快叫醒了 Totem 来干活。Totem 一本正经地拿着这个文件翻来覆去地看了半天，最后摇摇头说：这个……播不了。我一听就急了：你不是播放机嘛？在学校里你天天吹牛说你什么都能播，视频音频通吃，怎么关键时刻不行了？Totem 赶快解释：老大，别急别急，我说播不了，是有原因的。我播音频也好，视频也罢，都需要解码器。我现在手头没有这个 MP3 的解码器，所以不能播放。

有人会问：解码器是干什么的？要知道，音乐也好，视频也好，格式有很多种。就如同现实中看电影，有数字电影，就要用数字放映机；胶片电影，就得拿传统的放映机；在家里看光盘，就得拿 DVD 机；看录像带，就得拿录像机。听音乐也是，磁带的和 CD 的，肯定没法都塞进同一个机器里。Totem 这样的媒体播放软件就像一个电影放映员，解码器就是放映机。放映员再牛，也得有各种放映机做支持，没放映机他啥也放不了。

提示： Ubuntu 系统自带了一些格式的解码器，如 OGG 格式。但为避免版权问题而没有自带 MP3，RMVB 等解码器。

【安装 MP3 解码器】

我刚想质问 Totem 为什么不把解码器带全了，转念一想，我自己也没把语言支持包带全嘛。算了，还是不追究这个了，先说眼下怎么办吧。Totem 说：没关系，没带就让超级牛力去装嘛。当然，装软件这么重要的事情，我们得跟用户请示一下。于是，Totem 在屏幕上显示出了一条信息，就是图 3.35 所示这条。大概意思是，您要播放的这个东西所需要的解码器似乎没有安装，请问要不要现在去搜索一下需要装的解码器呢？

蜗牛同学还有什么选择吗？不听 Totem 的这 MP3 就播放不了啊，所以单击了“查找”按钮。Totem 在一番查找后，向用户汇报，说有这么几个包得装，装上就能播 MP3 啦，如图 3.36 所示。

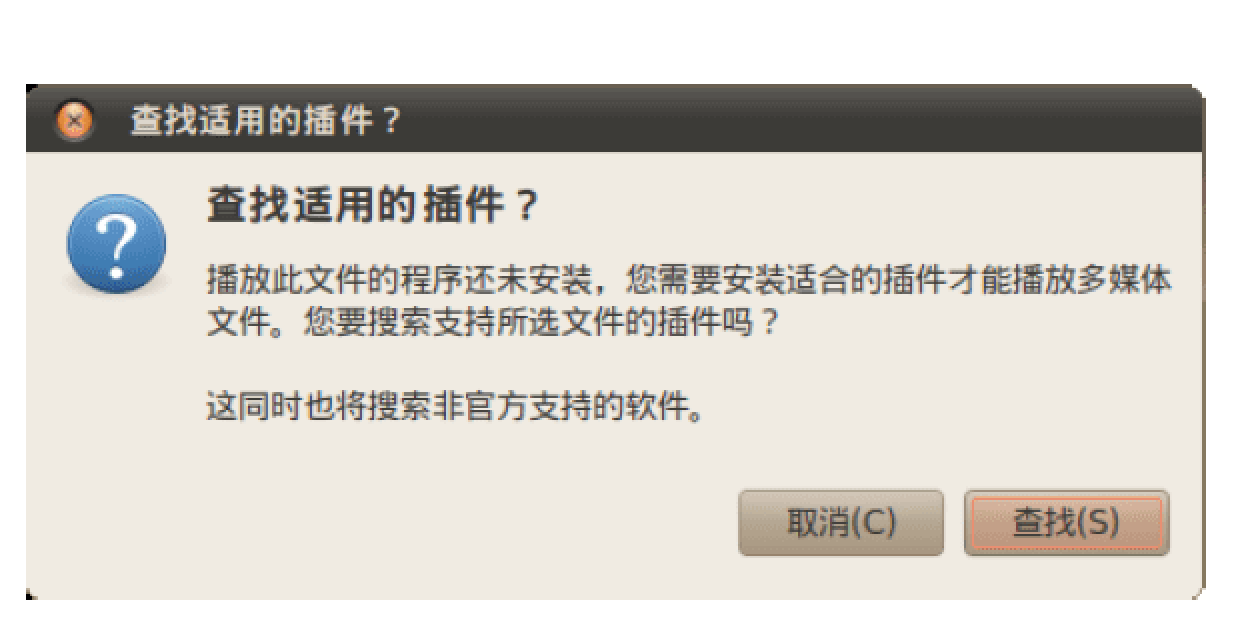




图 3.35　提示查找插件



图 3.36　Totem 搜索到的解码器

懒蜗牛同学当然单击了“安装”按钮。之后，估计您也猜到了，当然是超级牛力开始工作，装上了需要的解码器，最终 Totem 终于发出了久违的歌曲声。

不过懒蜗牛同学显然和我一样对这个音乐不感兴趣，却对 Totem 有了很大兴趣。

🔔提示：装好解码器后，将鼠标悬停在 mp3 文件上，几秒钟后可以预览文件内容。

【安装 rmvb 解码器】

以懒蜗牛同学以往的经验，系统刚刚装好，不能播放某音频或视频文件是正常的，比如以前的 Windows XP 系统刚装好时，没有软件可以播放 RMVB，必须得装软件，而装软件这个工作需要用户自己去做：上网找、下载，搞不好下载回来是个压缩包，还得先装解压缩软件。可是 Totem 竟然不用如此繁琐，直接提示缺什么东西，只要点个确定，该装的就都装上了，太人性化了。于是懒蜗牛同学很兴奋地又找来个 RMVB 文件双击了一下，Totem 出乎意料地没有再向用户报告什么，立刻换过视频解码器，开始播放 RMVB 的视频。

其实，这是因为在刚才安装 MP3 解码器的时候装的那个 gstream0.10-ffmpeg 软件包，包含了多种解码功能，其中就有对 RMVB 文件的支持。

3.4.2　安装 Mplayer 播放视频

虽然 Totem 能够播放不少类型的视频，不过毕竟他只是多媒体部门的小弟，要说播放器，那还得说 Mplayer 老爷子。Mplayer 可是多媒体部门的元老了，能力相当强，什么片都能放，什么 RMVB、FLV、AVI、WMV 全都不在话下（当然，前提还是得有解码器）。就算您没图形界面，人家在字符界面照样能依靠 framebuffer 给你放电影，甚至还能给您拿字符拼出电影看。现在时代发展了，都高清了，人家也不甘落后，照样能支持，什么硬解码软解码的，通吃。

🔔提示：如果需要 Mplayer 支持硬解码，首先要安装好显卡驱动，详见 3.5 节。

我们的懒蜗牛同学也在网上听说了 Mplayer 老先生的名声，于是叫来新立得，安装上了 Mplayer。装完了之后杯具了——这个软件装哪了啊？

【Linux 和 Windows 不同的处世哲学】

这里要介绍一下我们 Linux 和 Windows 7 的不同哲学了。我们两个的做事方式，对待用户的态度，以及很多观念都有很大的不同，因此导致了软件安装的位置也很有区别。

Windows 7 他们家族都比较小家子气，守住自己的一亩三分地不轻易让别人动。他不希望让别人了解自己的结构，所以 Windows 7 下的系统文件都是统一放在一个目录里面的。一般叫什么 Windows 之类的目录，别的软件谁也别想动。有什么样的领导，自然就有什么样的员工。Windows 7 下的其他软件也都学他那样，给自己建一个目录，一般在 Program Files 目录下，跟自己有关的东西就都放在那个目录里。软件之间泾渭分明，互不干涉，老死不相往来。顶多早晨上班见面点个头而已，很少有其他的交流，工作间里一片死气沉沉。

而我们 Linux 的世界就不同了。我们这里的目录是开放式的，按照文件的用途划分各种目录，而不是按照软件的名字来分。每个软件都把需要用的文件放在公共的地方，如果别的软件也需要用，甭客气，拿走。就比如刚才 Totem 装的那些解码器，其实就是一些解码用的库文件，所以放在了/usr/lib 下。现在来了 Mplayer，他要想播放视频也需要解码器，如果是在 Windows XP 下，那就请您自备一套，甭想用我的。就像 Windows Media Player 不能播放 RMVB 格式，装了 RealOne 后，RealOne 可以播放 RMVB，但是 Windows Media Player 照样不能，因为 RealOne 带来的解码器只能 RealOne 用。我们这里就不是这样，既

然有解码器了，就大家一起用。Mplayer 抄起 Totem 刚才装的解码器就能播放视频，这样，避免了同样功能的解码器重复开发，也省得为某一种视频格式专门装个软件。

🔔**提示：**由于软件之间相互共享资源的特点，Linux 下的软件更倾向于做得功能单一而强大。每个软件只专注于实现一个功能，需要复杂功能时通过多个软件的相互协作完成。

【软件安装位置】

说了这么半天，您一定着急了：“这 Mplayer 到底装哪了啊？”对于这个问题，我只好告诉你：“就像一锅被火车撞了的豆腐脑一样散落在各处了……”二进制文件放在了/usr/bin；库文件放在了/usr/lib；配置文件放在了/etc/；其他一些文档文件放在了/usr/share；还有一些数据放在了/var，乱吧。你肯定会说，这么乱，怎么管理啊？要删除的时候怎么办？这个问题的答案是：相信超级牛力！他会记得这些安装过的文件，删除的时候肯定一个不留。

🔔**提示：**删除软件使用命令：sudo apt-get remove <软件包名>来完成。

如果你求知欲很强，非要知道这个 Mplayer 到底都装了哪些文件，那也没问题。在新立得中搜到 Mplayer 这个软件包，右击这个包，选择“属性”，就弹出了属性窗口。这时候选择“已安装的文件”标签，就可以看到所有文件的去向了，如图 3.37 所示。

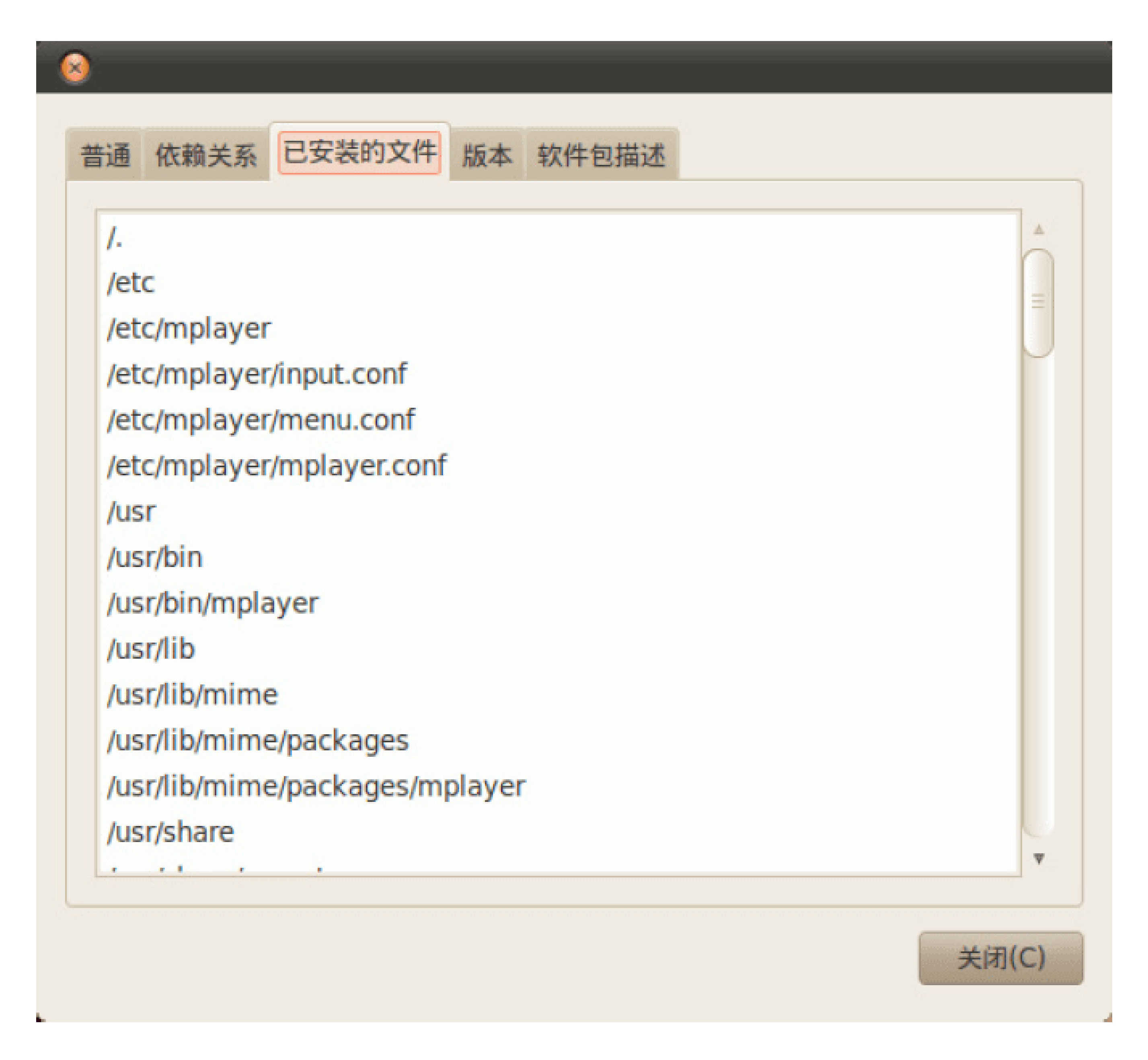


图 3.37　查看软件包所有文件

那么既然安装了 Mplayer，到底怎么运行他呢？如果是安装了一个图形界面的程序，我会按照这个程序的类型，自动把这个程序放在“应用程序”菜单里。不过 Mplayer 是个字符终端的程序，所以没有在图形界面体现。要想运行，打开终端输入：

```
$Mplayer /<path_to_file>/xxx.xxx
```

就可以运行了。这里，path_to_file 就是你要播放的文件的存储路径，xxx.xxx 就是你那个文件。这样运行了之后，就会看到 Mplayer 简洁的界面了，如图 3.38 所示。

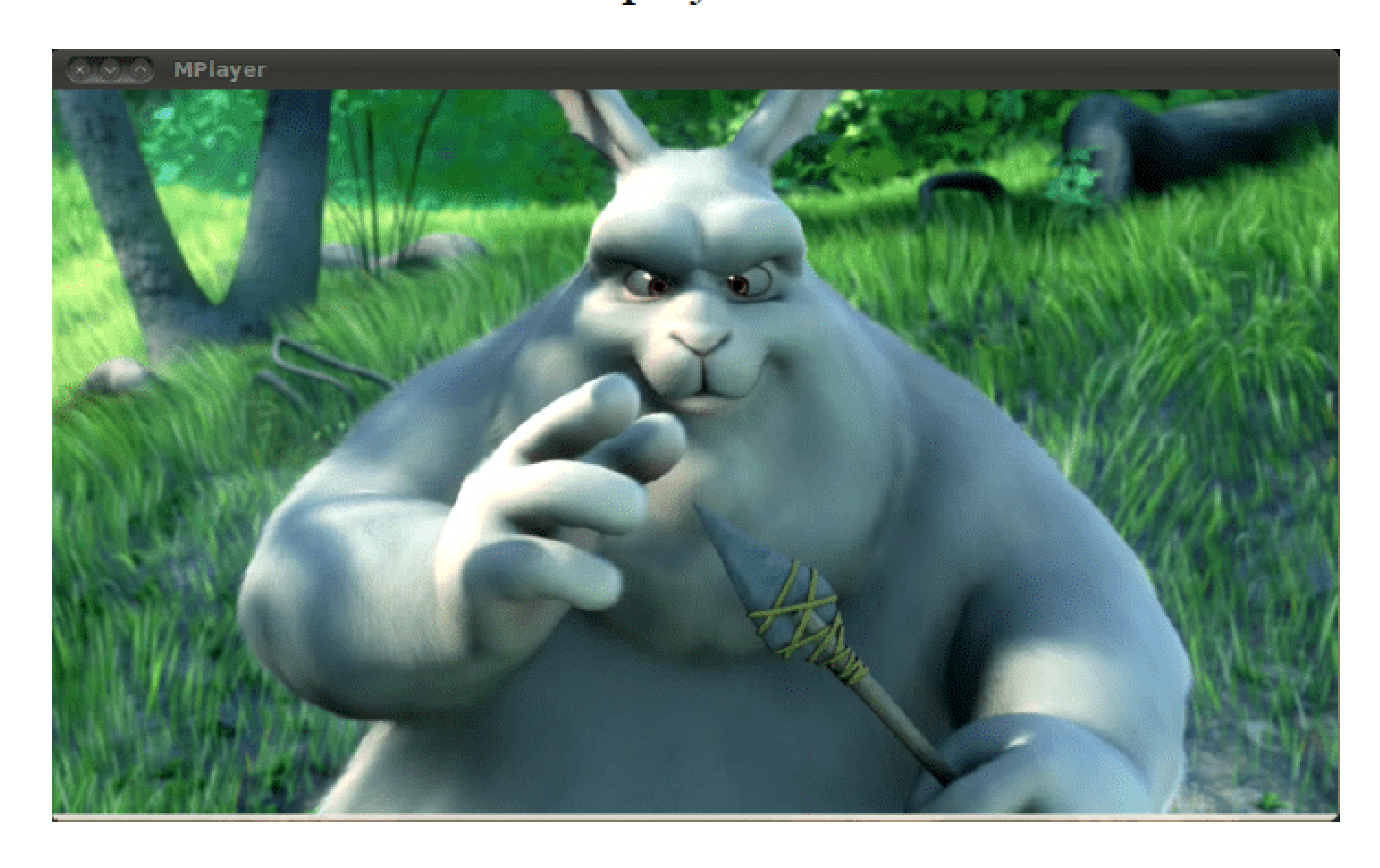


图 3.38　Mplayer 正在播放视频

不过这么运行当然有些麻烦，所以，懒蜗牛同学很快又叫来新立得，安装了 Gnome Mplayer。这又是什么呢？这是个 Mplayer 的图形化前端，可以像 Totem 一样操作，而后台还是 Mplayer 在做实际的播放工作。除了 Gnome Mplayer，还有 SMplayer、Kplayer 等很多 Mplayer 的图形前端。这些 player 当中，要数 SMplayer 功能最多，使用最方便。

3.4.3　播放音乐的 Rhythmbox

虽说 Mplayer 和 Totem 音频视频都是可以播放的，但毕竟视频才是主业，要说听歌，还是得找专业人士，比如我这里的 Rhythmbox 就是个不错的音频播放器。

Rhythm 是节奏，韵律的意思，所以我们就管 Rhythmbox 叫八音盒吧。八音盒可以播放各种音频文件（他也是依靠 Totem 装的那些解码器来播放的），还可以收听网络电台，还可以去 Ubuntu One 网络商店购买歌曲来收听，总之功能很强大。

提示： Ubuntu One 是由 Canonical 公司所推出的一项网络服务。该服务能够存储你的文件，并允许你在多台电脑上同步，还可以与好友分享这些文件。

懒蜗牛同学经过摸索，终于在“应用程序”|“影音”菜单中找到了 Rhythmbox 音乐播放器，赶紧把他叫起来干活，找来刚才那个 MM 发过来的文件（没办法，系统里暂时没有别的音频文件），让八音盒播放试试。这对于八音盒自然不是难题，马上，工作间里就到处咩咩声不绝于耳。

3.4.4　MP3 乱码

懒蜗牛同学使用八音盒播放音乐，并没有遇到任何问题。这要感谢支持八音盒的开源软件贡献者们。因为在不是很久的前一两年，那时候的八音盒会有 MP3 乱码的问题。这是怎么回事呢？

【乱码之源】

话说一个 MP3 的歌名、作者这些信息，是记录在 MP3 标签里面的。这个标签的记录方式有很多种，主要是这么几个标准：ID3v1、ID3v2 2.3、ID3v2 2.4、APEv2。这个 ID3v1 的标签支持 ISO-8859-1 编码，这是个国际通用的编码，但是很可惜，它是不支持中文的。

到 ID3v2 2.3 这个版本开始，增加了对 UTF-16 的支持，UTF-16 也是一套国际通用的编码，其中就包含中、日、韩等各国的文字了。到 ID3v2 2.4，更增加了对 UTF-8 的支持。UTF-8 和 UTF-16 都是国际通用的一套编码，所不同的是 UTF-8 以字节为编码单元，而 UTF-16 以双字节为编码单元，这样就有大小端问题（也就是哪个字节先传输的问题，不知道的可以 google 一下）。因此总的来说，UTF-8 相对更受欢迎一些。ID3v2 2.4 支持 UTF-8 编码，可并没有说一定要用，只是可以用 UTF-8，自然也可以用其他的编码。而 APEv2 标准就不同了，它规定了其编码必须统一为 UTF-8 编码。ISO-8859-1、UTF-8、UTF-16 都是国际标准的编码。

但 UTF-8、UTF-16 的出现都是后来的事情了，一开始，是没有国际标准的中文编码的，那时候只有国标——国家标准，也就是我们常见的 GB2312、GBK 和 GB18030。由于这些只是国家标准，所以开源软件的作者们（多数不是中国人）自然是忽略这些标准（当然了，每个国家都有自己的标准，听谁的啊？）。所以，如果使用 ID3v1 或者 ID3v2 类型，并且使用国家标准编码的 MP3 文件，就会出现乱码。如果是 ID3v1 或 ID3v2 类型，但是使用 UTF-8 或者 UTF-16 编码的，就不会出现乱码。如果是使用 APEv2 标准的 MP3 文件，就更不会出现乱码了，因为 APEv2 必须用 UTF-8 嘛（估计您已经听得满脑子乱码了吧）。不过遗憾的是很多播放软件不支持 APEv2 标准，好在我们的八音盒是支持的。

【解决乱码】

现在，我们 Ubuntu 系统的用户们是幸运的，基本不会遇到 MP3 乱码问题了。但是也不排除个别用户由于某种特殊需求，需要安装比较老的 Ubuntu 系统，那就会碰到 MP3 的乱码问题了。怎么解决呢？

经过前面的介绍我们知道了，MP3 之所以会出现乱码，就是因为编码不对，播放器不支持。那只要转换一下就好了。有个软件，叫做 mid3iconv，他就认识各种编码，让他把 MP3 的编码改成 UTF-8 的，就可以了。这个软件从哪里来？自然是让超级牛力请来：

```
$sudo apt-get install python-mutagen
```

或者让新立得安装也一样。装完了以后，就可以在你的家目录下运行：

```
$find . -iname "*.mp3" -execdir mid3iconv -e gbk {} \;
```

这样你的家目录下所有的 MP3 就都改过来了。

有人说，为什么 Windows XP 那里的播放器就没这么多问题呢？人家毕竟是商业化的产品嘛，而且是专门的“中文版”，自然得入乡随俗地支持国家标准编码。Windows XP 下的播放器默认有个叫 WMP 的，WMP 支持的 MP3 标签类型其实也不多，他不能支持 ID3v2 2.4 和 APEv2 的标签，还不如我们的八音盒支持的多。但是他很聪明，不支持的标签我就不显示，以显示那个 MP3 文件的文件名来代替，免得出现乱码，这一点倒是值得我们的软件们学习。

提示：这样转换后的 MP3 文件放到 Windows 环境下可能会出现乱码，因此不建议使用此方法更改 MP3 标签。推荐使用较新版本的、无乱码的播放器。

3.4.5　扩展阅读：开源和闭源

咱们说了 Windows 7 是个闭源的系统，而我是开源的系统。那么什么是闭源，什么又是开源呢？

闭源就是源代码不开放。我们知道，程序是程序员们一行一行地语句编出来的，C 语言也好，Java 也好，这一行一行的语句，就是这个程序的源代码。有了源代码，就能够百分之百地了解整个程序的构造，如何工作。而源代码是不能运行的，必须要把源代码变成可执行的二进制程序，这个过程叫做编译。源代码经过编译之后，才可以运行，但是编译之后的程序就不能够知道内部的构造了。我们平时在网上下载的各种程序，都是编译好的二进制程序，如果你想要它的源代码，对不起，不行！这是商业秘密，怎么能给你？给了你，我们的软件怎么卖钱？这种不开放源代码的程序，就叫闭源程序。

打个比方，就好像肯德基。香辣鸡翅谁都可以得到，只要花钱买就行，但是配方没人知道（虽然其实也没多好吃）。配方就相当于源代码，香辣鸡翅就相当于编译好的二进制程序，制作过程就相当于编译过程。如果有了配方（源代码），你就可以自己做香辣鸡翅（自己用源代码编译出二进制程序），甚至还可以根据口味对配方进行修改（根据自己的需求修改源程序，为软件增加自己需要的功能）。总结一下，画成图 3.39 所示这样，大概就好理解了。

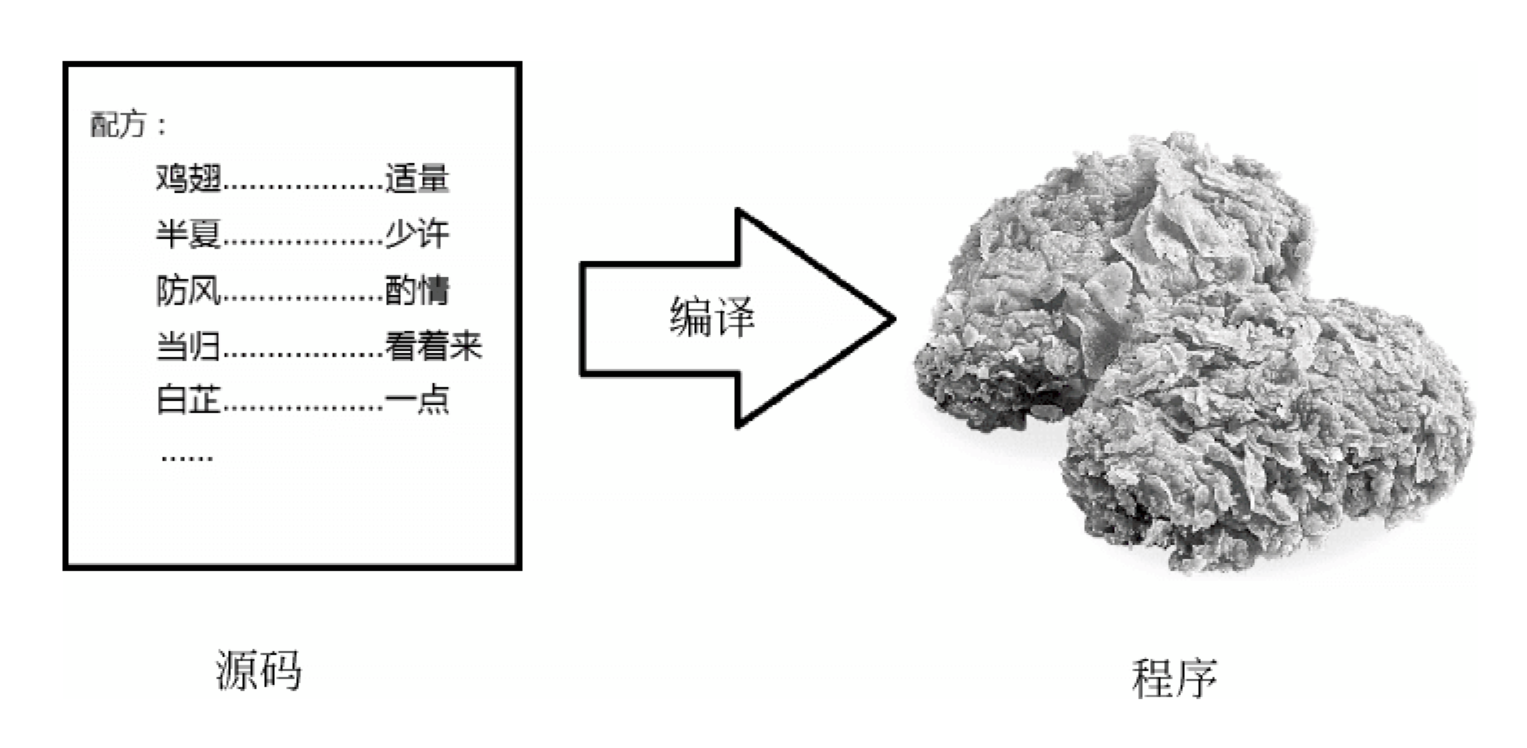


图 3.39　源码与目标程序的关系

那么开源又是什么？开源是一种精神，是乐于分享的理念。开源的程序就是你在获得程序的二进制文件的同时，还可以索取对应的源代码，从而学习这个程序的编写，或者完善这个程序，为这个程序添加功能。所有的一切都是对你开放的，而你要做的，就是如果你修改了这个程序，也同样要对别人开放。

再举个例子，有一天你发现，蒸鸡蛋羹的时候往里面加点牛奶，可以让鸡蛋羹更滑嫩。知道了这个窍门，你很高兴地把它告诉你的朋友，让他们分享你的经验，于是大家很高兴地也学会了做这样的鸡蛋羹。这就是开源。当然你也可能不把它告诉别人，而是保留这个秘密，甚至申请个专利，然后开个店去卖京城独一份的奶香滑嫩鸡蛋羹，这就是闭源。当然，这之中没有谁对谁错，谁好谁坏，只是理念不同而已。

3.5 安 全 软 件

常用的软件配置得差不多了，懒蜗牛同学根据使用 Windows 7 系统的经验，是该考虑系统的安全性的时候了。

3.5.1 杀毒软件

对于杀毒软件，我只想说一句话：我们 Ubuntu 系统真的没必要装杀毒软件。

3.5.2 防火墙软件

虽然作为一个桌面版的 Ubuntu 系统，被别人攻击的概率也不是很高，不过本着安全第一的原则，我们的懒蜗牛同学还是想要配置一下防火墙。

我们 Ubuntu 系统想用户之所想，急用户之所急，本着一切为人类服务的原则……嗯，好吧，自吹自擂的话就不说了，简单地说，我们自带了防火墙软件——ufw。

【不复杂的 ufw】

ufw 就是 Uncomplicated Firewall 的缩写，直接翻译过来就是不复杂防火墙。虽然是个命令行的软件，但使用起来确实不复杂。

（1）启动防火墙

系统默认情况下是不启用防火墙的，如果想要启用，只要运行以下命令：

```
$sudo ufw enable
```

这样防火墙就启动了，并且以后每次启动系统时，都会随系统启动。

（2）默认配置

光启动了还不行，防火墙都不知道该防点什么呢。这防火墙总得配置一下吧？当然，我们 Ubuntu 系统想用户之所想，急用户之所急，本着……长话短说，对于一般的桌面用户，只要运行以下命令使用默认配置就可以了。

```
$sudo ufw default deny
```

这样，防火墙会默认拒绝外部对本机的访问，允许本机对外部的任何请求。

（3）关闭防火墙

啥时候不想用防火墙了，运行这个命令：

```
$sudo ufw disable
```

这样防火墙就关闭了。和开启时一样，这个动作也会导致下次启动系统时防火墙依然不启动。

（4）查看状态

如果你忘了现在防火墙是开是关，就运行这个命令查看：

```
$sudo ufw status
```

这个就没什么可解释的了。

（5）开启/关闭指定端口

如果想对某个端口进行单独操作，比如想开启本机的 80 端口让别人访问，可以运行：

```
$sudo ufw allow 80
```

同理，如果要关闭外界对这个端口的访问，则运行：

```
$sudo ufw delete allow 80
```

（6）针对指定 IP 制定规则

如果看某个 IP 顺眼，想特殊照顾一下，允许那个 IP 访问本机，那么就这样：

```
$sudo ufw allow from 192.168.1.1
```

这个意思就是允许 192.168.1.1 对本机的访问。

【更加不复杂的 gufw】

上面说的仅仅是一些最基本的使用方法。如果你觉得这样用起来还不方便，没关系，我们 Ubuntu 系统想用户之所想，急用户……算了，再废话用户真该跟我急了。我们还有个软件，叫做 gufw，看名字就能猜出来，他是 ufw 的图形界面前端。默认系统中是没有的，需要叫超级牛力来安装：

```
$sudo apt-get install gufw
```

安装好之后运行 gufw 命令来启动这个软件，您就会看到图 3.40 所示的简单的界面。在这个界面上，可以简单地启动/停用防火墙、查看防火墙状态、添加规则等。功能和 ufw 是一样的。



图 3.40　gufw 界面

3.5.3 扩展阅读：为什么 Linux 不需要杀毒软件

好吧，我承认对于杀毒软件的陈述确实少了一点，大概不能够完全打消您心头的疑虑。好，那么我们就来说说，Linux 到底为什么不需要杀毒软件。

【病毒是有针对性的】

首先我们来了解一下病毒。病毒是什么？其实简单说，病毒只是一个程序，一个坏坏的程序。既然是程序，就跟其他的正常程序一样，依赖于不同的平台。啥意思？就是说，给 Windows 7 打工的，没法给我干活，给我干活的，也不理 Windows 7 那一套。我要是拎过一个 Windows 7 那边的程序跟他说，快起床干活，他压根听不懂，闭上眼睛继续睡，语言不通啊。所以，病毒也一样，针对 Windows 7 的病毒传染不了我，针对我们 Linux 的病毒也不可能传染 Windows 7。

【针对 Linux 的病毒】

那有没有针对 Linux 的病毒呢？答案是有的。

第一个 Linux 病毒诞生于 1996 年，澳大利亚的一个叫 VLAD 的组织用汇编语言编写了 Linux 系统下的第一个病毒：Staog，不过这个病毒只是个试验品，只是证明一下 Linux 也会感染病毒。这个病毒会感染二进制文件，获取 root 权限，然后说：Look！我获取了 root 权限耶。炫耀完了也就算了，并不做任何破坏性的事情。后来也有了一些有破坏性的病毒，但是数量很少。经过科学家计算，一个不装任何杀毒软件或防火墙的桌面用 Linux 系统，在互联网上裸奔并中毒的几率，大约比一个人花两块钱买彩票中五百万后立刻被雷劈中的概率大那么一点点（这是哪门子科学家算的）。病毒少，是 Linux 不容易中毒的一个原因。可为什么病毒少呢？

【没前途的 Linux 病毒】

话说有一个邪恶的人，出于某种邪恶的目的，想编个 Windows 病毒。他买书学习 Windows 的知识，找熟悉 Windows 的高人前辈们学习。经过种种努力，编出了一个病毒，然后把这个病毒放在自己的网站上，只要是使用 Windows 系统和 IE 浏览器上网的人一登录这个网站，就必定中毒。放上去之后，他等着，看着有 1000 人来到了他的网站，看着其中 900 多个纯洁的 Windows XP 系统感染了病毒(总有不用 IE 的，防护比较到位的 Windows 吧），他很有满足感，他觉得自己成为大牛了。

话说有另一个邪恶的人，出于某种邪恶的目的，他想编个 Linux 病毒。他买书学习 Linux 的知识——不过好像不太好找，好不容易找到基本也都是基础知识。找找高深的吧，还都是英文。好吧，英文的也看，对着字典慢慢研究。哦，对，还可以找找高人指导，不过……也不好找，找了半天找到一个高人，拜他为师吧。经过师父指点和自己的努力，他学到了很多 Linux 的知识。然后费尽心机编了一个 Linux 的病毒，把这个病毒放在自己的网站上，只要使用 Linux 系统，Firefox 浏览器上网的人一登录这个网站，就必定中毒。放上去之后，他等着，看着有 1000 人来到了他的网站——998 个人都是 Windows 系统……好吧，好歹还有俩用 Linux 的吧，可其中一个不用 Firefox，而是用 Opera。邪恶的家伙咬咬牙，忍！看最后一个——哈哈，这家伙是 Linux+Firefox，只要登录准中毒。可是只见着人来了转转又走了，一点事没有，临走还顺手改了自己的主页，上面写着：“小子，跟我玩你还嫩点。——师父留。”

通过对比我们得出结论——写 Linux 病毒，没前途！

【开源的本质带来的安全】

除了以上所说的原因以外，Linux 及周边软件的开源本质，也导致了病毒较少。

比如我，用户要装什么软件，都是叫 apt 去找。apt 可不是四处瞎找，而是去 Ubuntu 官方的软件源里去找——因为这些软件是开源的，所以可以随意拿来，放在一起，做成软件源，供 Ubuntu 们统一下载。官方的东西，自然没有病毒了，哪个娘也不能害自个孩子不是？

Windows 就不一样了，它上面的软件基本都是闭源的，要装，得自己上网搜，在某个网站搜到了，下载下来装。可这“某个网站”，就不知道靠不靠谱了。谁知道上面的软件有没有病毒呢？那么，微软的公司不能也开个官方的软件源，让大家都去他那下软件吗？当然不能了，都是闭源的软件，你拿来用都要给人家钱的（当然，也有免费的），拿来分发可能压根就是不允许的。

另一方面，Linux 的开源导致了大家都可以对其进行完善，一旦发现漏洞，随便谁都可以去修复这个漏洞，只要他有能力。可 Windows 呢？发现了漏洞，也只能漏着，等着微软公司去修。人家要是不修（比如可能正赶上食堂伙食不好导致的工人罢工），谁也没辙。

3.6　硬件和驱动

到目前为止，懒蜗牛同学的 Ubuntu 体验还算愉快，这主要是由于我带的驱动多，这个电脑上的硬件我都可以比较顺利地驱动起来。有人问，什么是驱动？

3.6.1　驱动——硬件的使用手册

电脑硬件，不像电视机电冰箱似的，买来插上就能用。硬件要想在计算机上工作，得需要会操作它的软件，这个软件，一般就是我们操作系统了。但我们操作系统，也不可能生来就会操作所有的硬件，就像你不是生来就会开飞机一样，得学、得考本、得移库、倒库、坡起、限制门。狐狸妹妹在旁边鄙视了我一下：“你见过飞机过限制门吗？！那是汽车。”反正，我们要想会操作一个硬件，也需要学习，这就需要驱动程序，任何硬件要想工作都是需要驱动程序的。

这时候可能有人会提出反对意见：“硬盘、光驱，这些也都是硬件，哪听说过要装驱动程序的？还有我的 U 盘、摄像头，也都是插上就能用，不用装驱动啊。”不用装驱动，不代表不需要驱动。硬盘光驱是最基本的存储设备，而且它们的驱动很简单，也统一。任何一家厂商生产的硬盘都是一样的用法，所以硬盘光驱的驱动就被集成在了 BIOS 和操作系统里面，不用额外安装。其他所谓不用装驱动的设备也一样，都是因为驱动集成在了系统里。比如 Windows 7 家以前的 Windows 98 系统，就不认识 U 盘，需要装驱动才行。到 Windows XP 这一代，就不用装了，集成了。到 Windows 7 这一代，就继承了更多的驱动。

提示： 由于操作系统存储于硬盘，硬盘必须先于操作系统工作起来，导致 BIOS 必须拥有操作硬盘的能力，因此 BIOS 必须包含硬盘驱动。光驱也是同理。

驱动就像一本给操作系统看的使用手册，上面写明了如何操作这个硬件，写哪个寄存器就把数据发出去了，从哪个寄存器读就把数据读回来了，往哪个寄存器写个什么数据就自爆了等（这是什么硬件啊……）。就像买来电视机，里面的使用手册一样。

针对不同的操作系统，需要有不同版本的驱动程序。这个好理解吧，因为我们是完全不同的系统嘛。像 Windows 7/XP/98，他们都是微软公司的系统，用的驱动还不同呢。我们这根本就不是一个阵营的，那就更不一样了。我们和 Windows 7 就像说着不同语言的不同国家的人。我们能看得懂的手册，Windows 7 看不懂，反过来也一样。你家电视机的说明书不也有中文版、英文版、韩文版、非洲土著语版吗。但是，并不是每个硬件厂家都给每个系统制作一份驱动，毕竟厂商人力财力有限。电视机也不是每台都有非洲土著语版的说明书嘛（压根就没有吧……）。

所以，一般硬件厂商会优先开发市场占有率最高的那个系统的驱动程序，哪个系统？目前来说，就是 Windows 7 家族的系统了。我们 Linux 就经常遇到一些因为厂家不提供驱动而无法使用的硬件，很多人还抱怨我们无能，冤枉啊……

3.6.2 安装受限驱动

其实现在我们 Linux 能够支持的硬件已经逐渐多起来，大多数主流的设备基本不用装驱动就可以使用了。一般像我们 Ubuntu 系统，装完了系统之后也就装装显卡驱动就可以了，没准连显卡驱动都不用装。

如果你的计算机上有什么硬件不能驱动起来，先看看能不能安装受限驱动。选择“系统”|“系统管理”|“硬件驱动”，打开之后就弹出如图 3.41 所示的窗口。



图 3.41　硬件驱动窗口

如果发现了可以驱动的硬件，就会在上方的列表框里显示出相应的驱动。如果想要安装，只要在上方列表框里选中相应驱动，并单击下面的“激活”按钮就可以了。

提示：“硬件驱动”中列出的是可以安装或已经安装的“受限驱动”，并不是所有的驱动都列在这里。所谓受限驱动，是指该驱动不开源，有版权，因此系统安装的时候并没有自动安装。

3.7 本章小结

咱们这回书中讲到，这个定居在硬盘里的兔子，慢慢找着工作的感觉了。也赶上这位懒蜗牛同学善于动手，把这个兔子配置得基本可用了。什么装软件的 apt、上网用的浏览器、各种聊天的工具、看片听歌的解码器，当然也少不了硬件驱动的程序，这些都配置好了，这个兔子就基本能够满足懒蜗牛同学的日常使用要求了。

然而论兔子的本事，远不止这些许的功能；那懒蜗牛同学，也未必满足于如此简单的使用。毕竟后事如何，还看下回分解。

第 4 章　我的系统我做主

经过了短暂的磨合，用户逐渐地熟悉了我这个 Ubuntu 操作系统，基本的上网、听音乐、看看视频之类的工作，都可以完成了。不过这自然还不算完，默认的外观，自带的软件，并不是每个人都用着顺手的。这不，懒蜗牛同学就开始着手把系统弄得更加个性化，更符合自己的习惯。

4.1　我 的 桌 面

有道是人凭衣服马凭鞍，操作系统看外观。外观对于一个操作系统来说也是挺重要的，所以，最先被懒蜗牛同学改造的，就是图形界面的样子。

4.1.1　默认桌面的配置

懒蜗牛同学看着我们默认的图形界面很不顺眼。为什么呢？因为窗口上的关闭、最大、最小化按钮竟然在左边，如图 4.1 所示。



图 4.1　默认的窗口布局

这导致每次懒蜗牛同学要关窗口的时候，鼠标都会划出一条很纠结的曲线——先向右上方去，发现按钮不在原来的位置之后又打轮拐向左上角来。听说我们这届的图形界面组那哥儿几个都是种苹果的出身，所以审美观点可能跟以往有些不同。不过没关系，我们 Linux 是为人类服务的，哪里看着不爽都可以改。

懒蜗牛同学上网学习了一下之后就开始动手了，过程并不复杂。

（1）首先启动命令行，并且运行 gconf-editor，或者按下 Alt+F2 键，输入 gconf-editor 也可以。运行起来如图 4.2 所示。



图 4.2　gconf-editor 界面

提示： gconf-editor 是一个 Gnome 桌面环境的配置编辑器，有点像 Windows 系统中的注册表。不过 Windows 的注册表是用来管理整个系统的，而 gconf-editor 只管 Gnome 桌面环境相关的设置。

（2）懒蜗牛同学依次展开了左侧的 apps|metacity|general 文件夹，这时在右侧上方窗口显示出了相关的键值，就是类似图 4.3 这样（以后咱就简单地说 xxx|xxx|xxx 文件夹中的 yyy 键值了啊）。



图 4.3　选中条目查看信息

（3）双击右侧上方列表框里面的 button_layout 键值，是个字符串类型的。把里面的内容改成这样：

```
menu:minimize,maximize,close
```

改后就像图 4.4 所示这样，然后单击“确定”按钮，窗口上的按钮马上就回归到图 4.5 所示的风格了。



图 4.4　修改键值

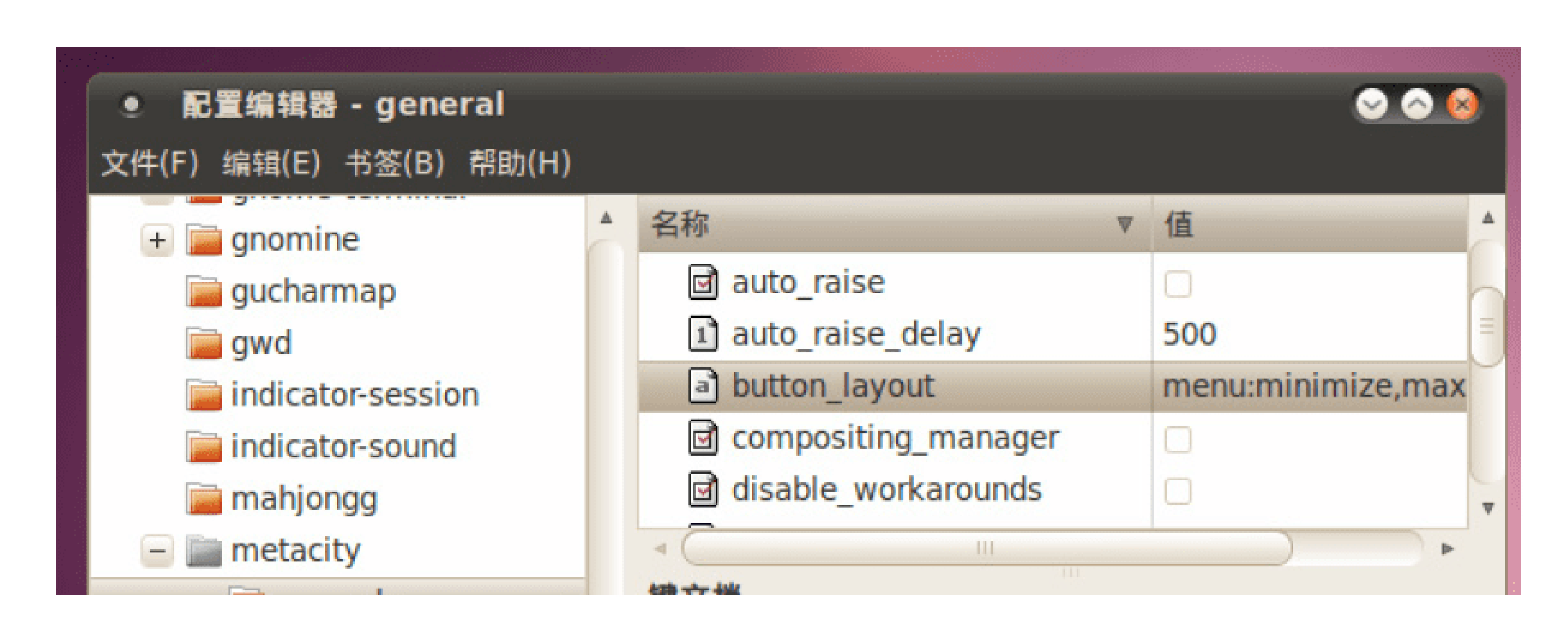


图 4.5　修改后的窗口布局

除此之外 gconf-editor 还可以设置很多东西，下面介绍一些常用的吧。

- apps|nautilus|desktop 目录中的 computer_icon_visible 键值，布尔型，用于控制是否在桌面上显示计算机。
- apps|nautilus|desktop 目录中的 trash_icon_visible 键值，布尔型，用于控制是否在桌面上显示回收站。
- apps|nautilus|desktop 目录中的 volumes_visible 键值，布尔型，用于控制是否在桌面上显示已挂载的设备。

提示： gconf-editor 的设置保存在~/.gconf/目录中，如果配置出现错误需要恢复默认设置，删除此目录即可。

窗口按钮顺手了之后，懒蜗牛同学又开始追求更加新奇的东西——3D 桌面。

4.1.2　3D 桌面的由来

这个 3D 桌面的概念，说起来是微软公司先想出来的。那时候微软公司正在研制 Vista，作为重要特性之一，Vista 系统支持三维桌面，当时还流传着一张图，展示了切换桌面窗口的时候以立体的形式展现每一个窗口，就是类似图 4.6 这个效果。这样新颖的设计着实让人们眼前一亮，也让 Linux 界的高手牛人们心头一热——这么个效果，我们也能做出来啊。

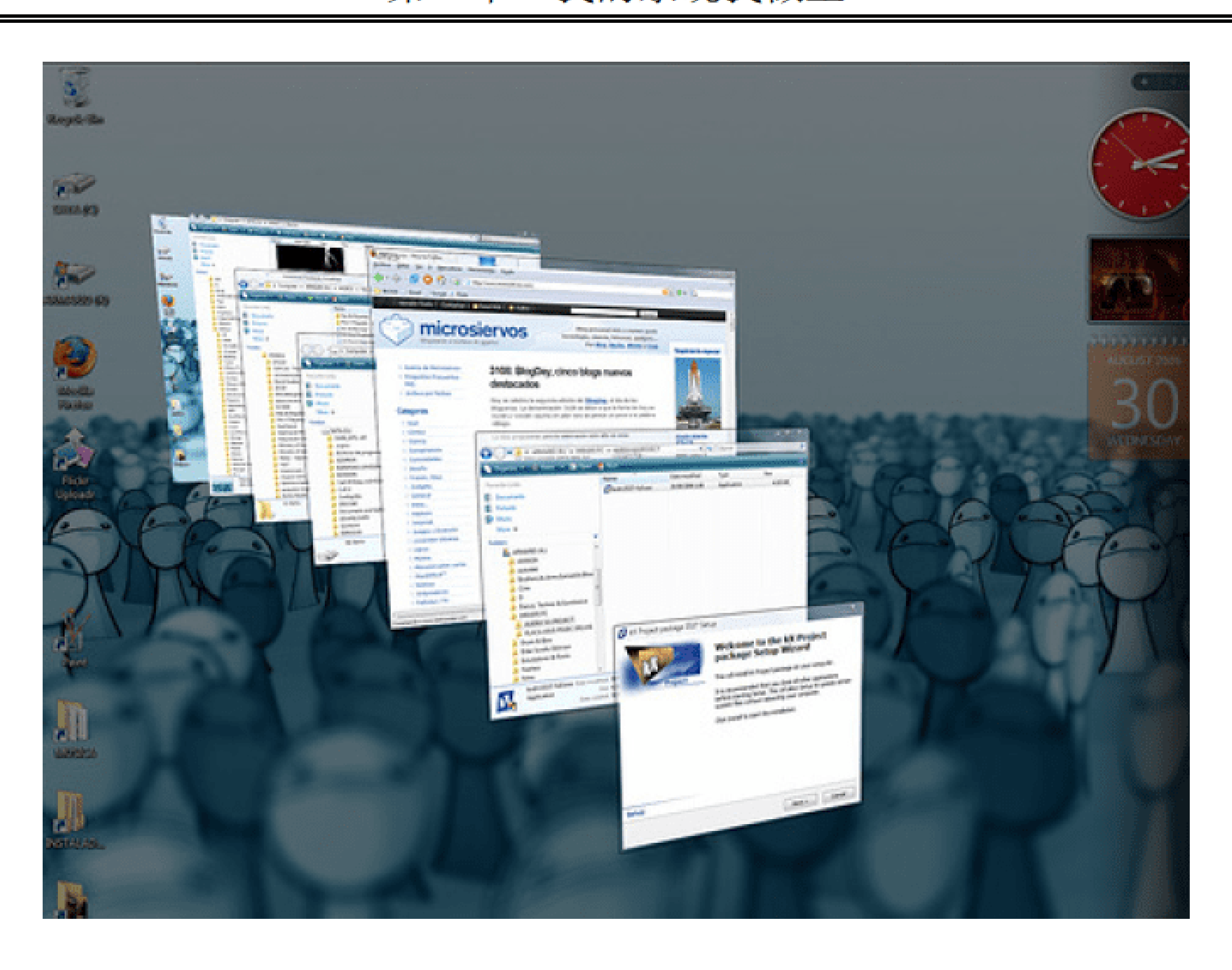


图 4.6　Vista 3d 桌面效果图

【重新发明的轮子——XGL】

要想实现 3D 的桌面，那么原来只能绘制 2D 图形的 Xorg 似乎就不能够胜任了。一小搓明白此真相的牛人们本着重新发明轮子的精神，设计了新的 X 协议实现，用于代替 xorg，这就是 XGL。有了 XGL 作为 X Sserver，加上 Opengl 的支持，上层的软件就可以比较轻松地绘制出立体的窗口，于是在此基础之上，Compiz 诞生了。

刚刚出世的 Compiz 功能不多，只能够让窗口随意透明，有拖动窗口时的果冻效果，以及最大化、最小化时的缩放效果，还有切换桌面时的立方体效果。但就是这些，已经足够让 Linux 的用户们大为惊艳，让 Windows 的用户们极度震撼。那时候 Vista 还在开发，还仅仅能够通过几张图片和几段视频来获知它所谓的 3D 桌面，而这时候 Linux 下已经可以看着一个硕大的立方体转来转去了，效果如图 4.7 所示。

提示： X 协议是 Linux 下的软件用来显示图形界面的协议，xorg 是实现 X 协议的服务端（即 X Server）。应用软件（即 X Client）通过 X 协议将要绘制的图形告诉 xorg，由 xorg 负责绘制在屏幕上。

【改造现有的轮子——Aiglx】

XGL+Compiz 的成功吸引了更多的人加入到完善 Linux 的 3D 桌面的工作中去。其中，Xorg 的用户和开发者们都在想一个问题——Xorg 为什么不能实现 3D 桌面呢？为什么非要换成 XGL？通过思考，最终他们得到了答案——哦，其实不用。

开源的优势这时候体现了出来，Xorg 是开源的，大家都可以来对他进行修改。于是有人为他设计了 Aiglx 插件，有了这个插件，Xorg 就可以和 XGL 一样实现 3D 桌面的绘制工作，甚至比 XGL 还要可靠。说来也是应该的，毕竟 Xorg 作为标准的 X Server 工作了那么多年了，多年的沉淀得到的稳定性不是一朝一夕可以追赶上的，带有 Aiglx 插件的 Xorg 逐渐替代了 XGL，到现在，Xorg 依旧是标准的默认的 X Server，要实现 3D 桌面，不再需要安装 XGL 了。

提示： 目前常见的 Linux 发行版中，只有 Suse 一直坚持用 XGL，其他发行版都已转向 Aiglx。

【分分合合的 Compiz】

在底层的 X Server 发展的时候，Compiz 也没有闲着，功能在逐渐完善。一方面更加稳定，效果更加流畅，另一方面更多新的效果被开发出来。可是渐渐地，追求稳定的开发者和追求效果的开发者分歧越来越大，最终导致 Compiz 项目组的两部分人分道扬镳。一部分人继续维护着 Compiz，不再增加什么绚丽的特效，而是注重稳定性和对资源的利用。另一部分人离开了 Compiz 项目，出去自立门户，成立了 Beryl 项目。这个项目旨在设计出炫目新颖的桌面特效，让 Windows 界的看看，他们想做的，我们一样能做出来，并且比他们做得好。从此，什么火焰、魔法、神灯、爆炸、屏幕涂鸦、飘飞雪花，各种特效层出不穷，真是千变万化，应接不暇。图 4.7 至图 4.9 所示是一些效果图。



图 4.7　立方体效果



图 4.8　火焰文字效果

然而有道是：天下大事，分久必合，合久必分。经过了一段时间的发展，Compiz 项目逐渐稳定，基本上没有什么 BUG 了，没什么目标的程序员们也开始想在 Compiz 里加入新的效果。而 Berly 在经过了创造的高峰期后，也没有什么更新奇的特效设计出来了。于是，两方人马不约而同地向着昔日的伙伴深情一望——咱还合并吧。结果，神机百变的 Beryl

和老成练达的 Compiz 终于又走到了一起，两个项目合并为了 Compiz Fusion——也就是现在我们 Ubuntu 系统中用来实现 3D 桌面效果的软件。

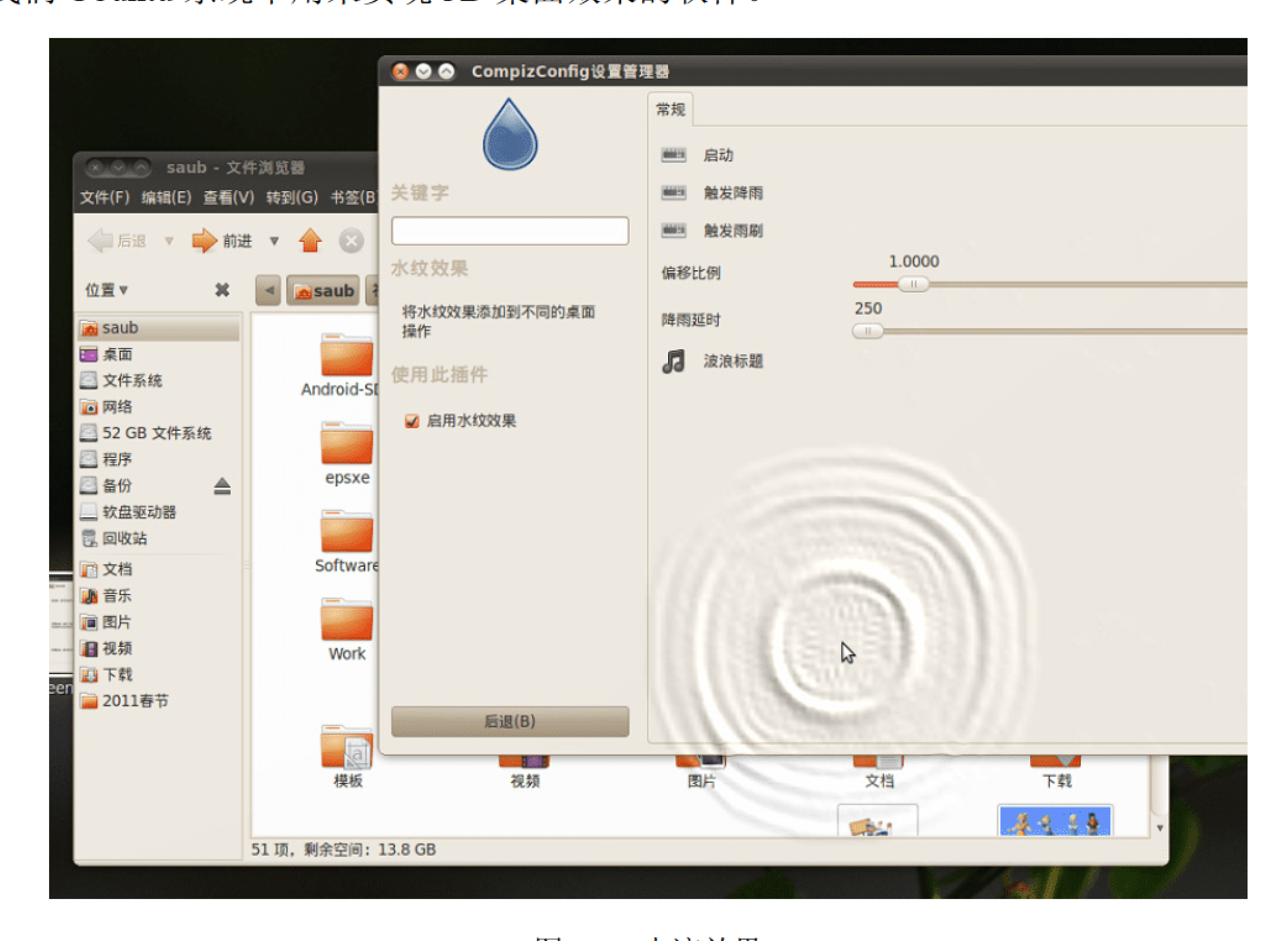


图 4.9　水滴效果

4.1.3　体验 3D 桌面

其实在 3D 桌面非常成熟的今天，我们 Ubuntu 已经默认自带一些简单的效果，大约就是当年 Compiz 最初时期的那些。如果想要体验，首先要安装好能够支持 Opengl 的显卡驱动，之后只要单击“系统”|“首选项”|“外观”里面的“视觉效果”标签，像图 4.10 这样选择“正常”或者“扩展”单选按钮，就可以看到了。

提示：如果“正常”和“扩展”单选按钮都无法选择，请先检查显卡驱动安装是否正确。

扩展里面的效果自然要比正常多一点，不过也很有限。如果想有更多的效果，并且可以自己配置，那么需要像懒蜗牛同学这样，安装 Compiz 设置管理器。安装也简单，叫超级牛力：

```
sudo apt-get install compizconfig-settings-manager
```

就可以了，或者让新立得装当然也行。装好之后就可以在“系统”|“首选项”里面，看到“Compizconfig 设置管理器”。运行之后，就可以看到类似图 4.11 这样的界面。

在这里有很多效果可以选择，单击感兴趣的内容后，会有详细的说明和快捷键的设置，这里就不多说了。总之，用户通过这个 CompizConfig 设置管理器就可以根据自己的喜好搭建自己的桌面效果。



图 4.10　设置视觉效果



图 4.11　CompizConfig 设置管理器

4.1.4　扩展阅读：Xorg

本节在咱们说 3D 桌面时提到了 Xorg，那么 Xorg 是个什么软件呢？

【Xorg 是谁】

话说我们 Linux 系统里，系统跟图形界面是分开的。绘制图形界面的事情由专门的图形部门来负责。而图形部门的老大，就是 Xorg。他会跟硬件打交道，会用显卡（当然，用显卡也得经过我），能在显示器上画东西，想画什么画什么，谁要想显示点东西给用户看，都得经过他。

【X 协议】

要想跟 Xorg 打交道，在显示器上显示出图形来，得懂他们图形部门的黑话——学名叫协议。他们说话使用一种叫做 X 的协议。一个程序通过 X 协议告诉 Xorg 要画什么，那么这个程序就是 X Client，而负责在屏幕上画的 Xorg 就是 X Server。反正要想显示图形，就得用这种黑话跟 Xorg 去说。每一个要显示图形的程序都得会这种黑话，比如狐狸妹妹（以下“狐狸妹妹”或“狐狸”特指 Firefox），要显示东西，就说：“驼子碗，筛土的抛闪！”那意思可能就是说画一只猪。当然，这就是打比方，其实我可不懂他们的黑话（这一点不像 Windows 7，他本身兼职负责画图形）。

那么程序要画什么直接跟 Xorg 说就行了么？其实也行，比如 Mplayer，他可以直接跟底层的 X Server 交流，可那就像是在字符界面下看片了——没有窗口，图像没法移动，没法全屏，没法最小化。因为 MPlayer 只负责放片，像画窗口、移动窗口什么的这些事情他可不管。那谁来管呢？这时就需要一个窗口管理器。

【窗口管理器】

我们这里默认的窗口管理器叫做 Metacity（就是 Gnome 下的默认窗口管理器）。程序要画什么东西其实是跟他说的，不直接跟 Xorg 说。比如 Mplayer 说：“画一只猪。”（当然是用 X 黑话）。于是 Metacity 转头告诉 Xorg：“在某某位置画个方的窗口，在里面画一只猪。”过一会儿可能用户觉得 Mplayer 的窗口挡着他和 MM 聊天了（那是，猪哪有 MM 好看呀），就把 Mplayer 的窗口挪了挪，于是 Metacity 又对 Xorg 说：“把刚才那只猪和窗口往左移动 3.2 厘米。”这个过程 Mplayer 是不知道的，他只管专心地向 Metacity 描绘着影片中的一幅幅图像：“猪、走路的猪、跑动的猪、跌倒的猪、捆绑的猪、烤熟的猪……”

提示：如果安装了 Compiz，则 Gnome 界面将使用 Compiz 作为窗口管理器。

4.2　我的网络世界

懒蜗牛同学安装了 Compiz Fusion，打开了 3D 特效，桌面看上去比以前好看了不少。接下来，懒蜗牛同学又开始挑选合适的网络相关的软件了。

4.2.1　满身插件的狐狸妹妹

先说上网用的浏览器吧。懒蜗牛同学虽然配置过了狐狸妹妹，但还是觉得用起来不大顺手。因为他以前在 Windows 7 那里是用一个叫做 Chrome 的浏览器来上网的，那个浏览器简洁方便速度快，价格便宜量又足，懒蜗牛同学一直都用他，忽然一换成狐狸妹妹，还是有点不大习惯。

【安装扩展，实现功能】

既然不习惯那就换呗。好在这个 Chrome 浏览器也有 Linux 版。于是懒蜗牛同学就牵着狐狸去网上找到了 Chrome 的 Linux 版，并且开始下载。狐狸妹妹接到指令，就开启了一个下载进程，像图 4.12 这样不急不忙地下载着。



图 4.12　Firefox 默认单线程下载

懒蜗牛同学看着有些不爽了，想当年在 Windows 7 的时候，下载起来有雷有车，那速度多快啊。如今到此地界，难道就只能靠这浏览器自带的下载功能？他赶紧去网上问了问，这才得知，这火狐狸本身虽然只是个浏览器，但是可以安装很多强大的扩展程序，装了不同的扩展，就有了各种不同的功能，其中就包括多线程下载的扩展。懒蜗牛同学对此很感兴趣，于是赶紧先去找个下载用的扩展来装装。

注意： 下载 Chrome 时应选择 deb 格式的安装包。

给狐狸妹妹装扩展很简单，跟用新立得装软件差不多，也是“超市化”安装。

（1）打开狐狸妹妹后，单击“工具”|“附加组件”，就会看到安装附加组件的窗口，如图 4.13 所示。



图 4.13　附加组件窗口

提示： 所谓附加组件，包括扩展、插件、主题和语言。

（2）要装什么扩展，直接在上面的文本框里面输入扩展的名字，然后单击文本框右侧的那个放大镜开始查找。

（3）找到的扩展就会出现在下面的列表框里，单击“添加至 Firefox”按钮就可以开始安装了，如图 4.14 所示。



图 4.14　搜索并安装附加组件

当然你可能不知道到底要装什么插件，也不知道都有什么插件，没关系，单击右上角那个“浏览全部附加组件”链接。单击之后会打开一个网页，其实就是这个地址：https://addons.mozilla.org/zh-CN/firefox/。在这上面分门别类地介绍了现在流行的，好用的，各种各样的组件。看到合适的插件，如果想要安装，直接单击网页上的“添加到 Firefox”按钮就可以装上了，类似图 4.15 所示这样，很方便。



图 4.15　通过网页安装扩展

【Downthemall 扩展】

懒蜗牛同学之前已经通过在论坛询问得知狐狸有一个扩展叫做 DownThemAll，是用来下载的，于是就直接搜索到这个扩展，告诉狐狸妹妹“装！”狐狸妹妹赶紧去下载这个扩展，下载下来之后把这个扩展保存在硬盘里属于她的那块空间中，然后告诉懒蜗牛同学：“重启我一下就可以用啦。”懒蜗牛同学点点头：“好，重启吧。”

于是狐狸妹妹回硬盘，收拾了一下，再次启动来到内存——这回她兜里就插上了 DownThemAll 扩展了。现在用户再单击任何下载链接，就可以像图 4.16 这样选择用 DownThemAll 下载了。

这样的好处主要是可以多线程下载，提高下载速度。但是这只限于普通的 HTTP 或者 FTP 下载。什么迅雷专用链接，快车专用地址之类的自然就没戏了，BT 电驴啥的也是由其他的专门软件来负责的，DownThemAll 只负责最基本的下载。不过目前来说，这就是懒蜗牛同学想要的，足够了。

装上了 DownThemAll 扩展，懒蜗牛同学感觉这 Firefox 还不错，于是又在网站上面浏

览其他的扩展软件。这里的扩展软件还真是琳琅满目，干什么的都有。有用来预报天气的，有方便管理 blog 的，有网页开发相关的，有阅读新闻的，有下载的……真是没有做不到，只有想不到。



图 4.16　安装 DownThemAll 插件后可选择用其下载

【天气预报的扩展】

懒蜗牛同学先是看到了一个叫做 1-ClickWeather 的扩展，这是个天气预报插件，懒蜗牛同学一想，这浏览器带天气预报，挺有意思，就顺手在网页上点了旁边的“添加到 Firefox”。然后自然是狐狸妹妹下载这个扩展，把这个扩展保存在硬盘里属于她的那块空间中，然后再告诉懒蜗牛同学：“重启我一下就可以用啦。”懒蜗牛同学又点点头：“好，重启吧。”

于是狐狸妹妹还是回硬盘，收拾了一下，又一次启动来到内存——这回她兜里就插着 DownThemAll 和 1-ClickWeather 这两个扩展了。好在扩展一般都不大，几百 KB 而已，揣在狐狸身上也不沉。这回狐狸刚一启动，就先弹出了一个用于设置 1-ClickWeather 的窗口，如图 4.17 所示。



图 4.17　1-ClickWeather 设置页面

因为是天气预报，世界这么大，不能都给你报，你得选择一下你要看哪里的天气预报。懒蜗牛同学按照图 4.17 那样进行了一下设置，设置好之后单击 Save&Exit 按钮。然后，狐狸就正常启动了，启动后就可以在右下角看到最新的天气预报，如图 4.18 所示。

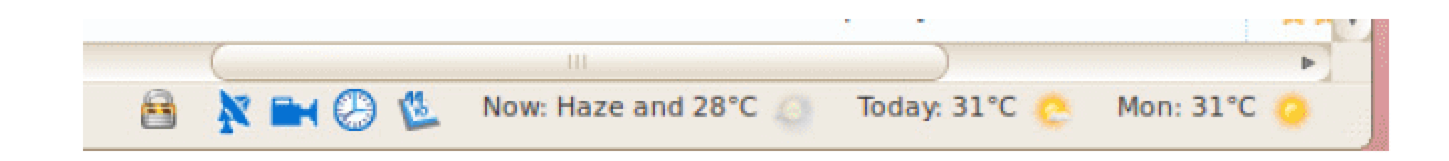


图 4.18　装好后在 Firefox 状态栏看到天气状态

【保存 Flash 的扩展】

除此之外，懒蜗牛同学还看上了 DownloadHelper 扩展。这个扩展对他也很有用，平时他经常上一些地瓜啦，马铃薯啦之类的网站，看些有意思的视频。不过这些网站都是在线看的，不提供下载。懒蜗牛同学的网速又不是很快，再说，就算网速足够快，也不能保证什么时候想看就能上去看，今天还有的视频，明天就可能被大螃蟹咔嚓走了。所以懒蜗牛同学希望能够把网页上的视频保存下载，一看到这个 DownloadHelper 扩展正好就是干这个的，赶快让狐狸妹妹去装。

提示：DownloadHelper 只用于下载网页中 Flash 形式的视频，其他形式无法下载。

狐狸妹妹二话不说，责无旁贷，赶紧下载、保存、重启、揣着 3 个扩展进内存，一气呵成。这回狐狸妹妹的界面上多了 DownloadHelper 的按钮，就是图 4.19 所示这个位置。以后如果网页上再有什么 Flash 的视频、音频，狐狸妹妹就会通过 DownloadHelper 这个扩展检测到，并且通知用户，有个视频或音频可以下载。如果用户需要，就可以把这个视频下载到本地硬盘上了。



图 4.19　Firefox 界面上的 DownloadHelper 按钮

这之后，懒蜗牛同学又装了七八个有意思的扩展。装了这一大堆扩展后，虽然把狐狸妹妹累得不轻，启动都变慢了，但懒蜗牛同学却开始对狐狸妹妹有好感了。因为他体会到了狐狸妹妹最大的特点——定制，就是可以变成你想让她变成的任何样子，按照你的口味，打扮起来给你看。不过有时候也被人诟病说这么多扩展拖慢了速度，这方面确实不如 Chrome。哦，对了，懒蜗牛同学那 Chrome 早就下完了，狐狸妹妹也提示过他了，他到底装不装啊？

4.2.2　会分身的 Chrome

懒蜗牛同学和狐狸妹妹又缠绵了一阵子，才终于想起来——哦，对了，我是想装 Chrome 的，明明是要换用自己顺手的浏览器，怎么差点被狐狸迷惑了。他赶紧找到下载的 Chrome 的 deb 包。由于懒蜗牛同学没有更改保存地址，狐狸妹妹就把下载的东西存在了懒蜗牛同学的家目录下的“下载”目录。

提示：所谓家目录，就是/home 目录下以用户名命名的那个目录。比如 lanwoniu 用户的家目录就是/home/lanwoniu。每个用户只能在自己的这个家目录下进行操作，其他的目录没有权限。

提示：Linux 中可以使用“~”符号代表家目录。例如，对于 user1 用户，运行 ls ~就相当于运行 ls /home/user1，而对于 user2 用户则相当于运行 ls /home/user2。

懒蜗牛同学找到了 deb 文件，双击安装。不一会，超级牛力就按照操作流程解开了这个 deb 包，从里面拎出一只浏览器。

【用多进程追求速度的极致】

这家伙长得色彩斑斓，红黄绿蓝四色相间，样子简简单单，倒也清爽。大家没来得及仔细看，这家伙就被超级牛力带进硬盘里安排住处去了。狐狸妹妹有些不高兴地说：“哼，回头倒要看看他有什么本事。”是啊，这家伙来了自然就会抢狐狸的饭碗，难怪狐狸不高兴。

没过一会儿，Chrome 就被懒蜗牛同学叫起来干活了。Chrome 一跑进内存，一下子变成了 4 个 Chrome！也就是分出 4 个进程。他们在工作间里窜来窜去，那叫一个热闹。4 个 Chrome 分别忙着各自的工作，有的负责向图形界面申请窗口；有的负责在窗口上绘制各种标签和按钮；有的打开网口，访问默认的主页。虽然这么多进程挺闹腾，倒也是各司其职，有条不紊。而且这样一来，这些事情是同时进行的，效率就会高一些。很快，Chrome 就为用户显示出了一个简洁的界面，如图 4.20 所示。

提示：可以用 ps –A 命令来查看当前运行的进程。



图 4.20 Chrome 运行界面

相比之下我们的狐狸妹妹就比较本分，一步一步进行。先向图形部门申请绘制窗口，窗口批下来之后再在窗口中绘制好各种文字、菜单、按钮之类的。这之后再根据用户设置的主页，上网申请数据，数据来了再显示给用户。如果有插件，在这之前还要加载各种插件，难怪有人觉得狐狸有点慢呢。

这还不算完，等到懒蜗牛同学真的开始使用 Chrome 浏览网页我才知道，原来这 Chrome

不只是 4 个，随着打开的网页越来越多，Chrome 不断地复制自己，工作间里的 Chrome 进程也越来越多。好家伙，转眼间工作间里就看不见别人了，到处都是 Chrome。不过好在他们每个占用的资源都不多，因此也不会给别人带来什么麻烦，这也算是 Chrome 这家伙的特色之一吧。

【争夺默认浏览器的地位】

Chrome 来了以后，这工作间里面就开始不太平了。狐狸妹妹和他谁也不服谁，整天吵架。系统中可以有很多浏览器，但是默认的浏览器只有一个。原本默认浏览器当然是狐狸妹妹，可当 Chrome 启动的时候就显示出图 4.21 所示的这个界面来告诉用户：我不是默认浏览器，把我设成默认浏览器吧。

如果用户把 Chrome 设成默认浏览器，狐狸妹妹启动时也会向懒蜗牛同学抱怨：“以前我是默认的，干得好好的怎么就换了呢？您还是给我设回来吧。”也搭上我们的懒蜗牛同学没准主意，墙头草，随风倒。今天设狐狸为默认的，明天改 Chrome 是默认的。

提示： 可以手动设置默认浏览器，单击“系统”|“首选项”|“首选应用程序”，选择 Internet 标签，如图 4.22 所示，在上方“Web 浏览器”对应的下拉列表框中选择即可。

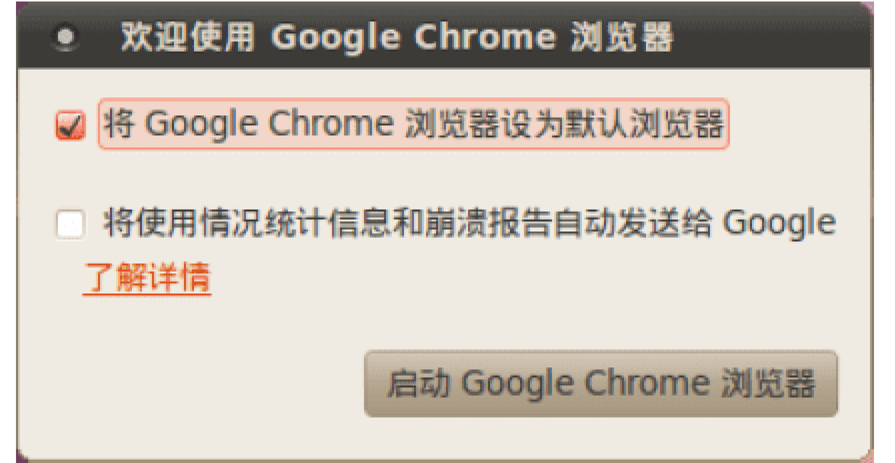


图 4.21　Chrome 询问是否设置为默认浏览器



图 4.22　设置默认浏览器

不过最终懒蜗牛同学还是确定了狐狸妹妹的默认浏览器地位，毕竟是我的原班人马嘛。别的不说，由于狐狸是软件源里的软件，所以如果有个什么版本升级 BUG 修复之类的，都是自动的。那时超级牛力（以下“超级牛力”特指 apt）会接到 Canonical 学校发来的通知：狐狸同志，工作一向兢兢业业，刻苦提高技术水平，今经组织决定，晋升狐狸为 x.xx 版本，建议进行升级。得到用户的同意之后，超级牛力就去下载来新的部件，把我们机器里的狐狸按照学校提供的步骤改装成新的狐狸。

我带来的每一个软件和通过超级牛力安装的软件都有这种待遇，而像 Chrome 这种从网站上下载来安装的软件就不行了，要想升级，只能再去网上下载新的版本，自己重新安装。

提示：软件源里有 Chromium 浏览器，算是 Chrome 的试验版。如果喜欢 Chrome 风格但又希望它能跟随系统进行更新，可以安装源里面的 chromium-browser 这个软件包。

所以 Chrome 们整天不服气，跟狐狸说："你也就是因为有靠山，那笨兔子内核跟你一伙的，所以才能容得你这么飞扬跋扈。"狐狸自然也不肯罢休："你那样子又有什么好了，再说了，要变成你那样子简单得很。给我装上 Chromifox Basic 主题，再装上 Total Rechrome 扩展倒置标签兰和地址栏，最后装个 Hid Menubar 扩展隐藏掉菜单栏，跟你也没什么区别。只不过用户不愿把我打扮得像你那么难看罢了。"另一只 Chrome 反唇相讥："哼，变成我们这样有什么用，装那么多插件扩展的，臃肿啊，哪像我们简洁高效。"狐狸怒道："你想装这么多插件还没人给你开发呢！"

提示：目前最新版的 Firefox 默认界面已经逐渐向 Chrome 学习，不需要安装什么扩展也很精简。

4.2.3 干净利索的 Opera

整天看着 Chrome 和 Firefox 这么闹腾也不是个事儿，于是熟读三国的懒蜗牛同学想到了一个好办法——他又装了一个浏览器，Opera。

【认识 Opera】

Opera 小姐来自挪威，身材比狐狸妹妹好些，体态轻盈，长相倒是一般，就是图 4.23 这样。她总爱穿着一身大红的衣服走来走去，举手投足间透露出一种高贵的气质，而她最大的特点，就是干活麻利。

提示：Opera 浏览器也不在软件源中，需要到官方网站 http://www.opera.com 下载。



图 4.23　Opera 运行界面

作为一个浏览器，最基本的技能就是渲染网页，所谓渲染网页，就是根据从网站上取回来的 HTML 语言的描述，把相应的文字和图片及 Flash 的元素显示在页面上。所以浏览器每天的主要工作就是解释 HTML 及 JSP 等语言，并显示出漂亮的页面。

Opera 在解释语言上有她自己的一套办法，解释起来迅速快捷。具体为什么快，没人知道，因为她虽然也免费，却不像狐狸妹妹和 Chrome 一样来自开源社区。Opera 是一个闭源的软件，我们不能够了解她的内部构造。除了速度以外，Opera 也有扩展和插件，但由于她不开源，她的扩展都是由她的东家——Opera 软件公司设计的，不像狐狸妹妹的扩展都是广大热心的网友们写的，所以 Opera 的插件没有那么多。

另外，Opera 还是个多才多艺的浏览器，除了能做浏览器，她还能 irc 聊天，能作为邮件客户端，现在她又学了个新本事，叫 Unite，能建立简单的 HTTP 服务器用来跟别人共享文件了。虽然狐狸妹妹通过插件也可以实现一些功能，但毕竟跟 Opera 差了那么一点。

提示： 本质上 Chrome 并不开源，开源的是 Chromium，Chromium 是 Chrome 的开源实验版。各种新的特性和技术会先在 Chromium 上应用。

【三足鼎立】

这样一来，3 个浏览器成三足鼎立之势，任何两个浏览器都可能联合起来对付第 3 个。Chrome 发飙的时候，狐狸妹妹会和 Opera 姐姐联合。因为 Chrome 一分就好多个，人多势众，而狐狸和 Opera 都是单进程的，团结起来才是办法。狐狸要犯脾气的话 Chrome 和 Opera 会联手。因为狐狸是我的嫡系软件，跟我坐同一张光盘来的，还能自动升级，跟其他软件的关系也比较密切。Chrome 和 Opera 则都是外来的，手动安装的，初来乍到还人生地不熟，自然要结成团体。Opera 使小姐性子的时候狐狸和 Chrome 肯定站在一边，因为这俩都来自开源项目，大家的思想和理念比较一致。Opera 的闭源思维难免和他们会有格格不入的地方。于是 3 个浏览器互相牵制，工作间里面从此就天下太平了。

4.2.4　更多的浏览器

后来，懒蜗牛同学又先后安装了很多浏览器，有极其简约的 Uzbl，有轻巧的 Epiphany，有简陋的 dillo，甚至纯字符界面的 lynx 和 links 都装上了。他装这么多浏览器干啥？一块运行起来看热闹玩？当然不是，他是在体验，在选择，寻找最适合自己的那一款浏览器。

我们 Linux 世界里的软件总是多种多样的，同样功能的软件可能会有很多款，而且特色各异。每个人都可以找到适合自己的那一款。甚至如果没有哪款合你心意的，还可以通过修改某个浏览器的代码（因为多是开源的嘛），来创造出你自己喜欢的浏览器来。比如狐狸妹妹深受广大用户喜爱，然而就是干活速度有些慢，于是有人就在狐狸代码的基础上加以改进，出现了疯狐狸（MadFox）浏览器。这就是 Linux 的世界，在这个世界里，人类才是软件的主人。

4.2.5　BT 下载软件大选秀

虽然懒蜗牛同学有了狐狸妹妹的 DownThemAll 插件，但是毕竟功能有限，网络上很

多的资源都是需要 BT 下载的，于是懒蜗牛同学在选择了适合自己的浏览器之后，又开始了 BT 软件的海选。

【比赛规则】

懒蜗牛同学的海选工作主要通过网络进行，由狐狸妹妹和狗狗哥作为星探，到世界各地打探 BT 人才，并到懒蜗牛处报名登记。之后经过懒蜗牛的初步筛选，由超级牛力负责召集获得参赛资格的软件前来比赛，超级牛力力所不及的（也就是不在源里的啦），就由狐狸妹妹去邀请。把 BT 软件们找来之后，进行初赛，由 3 位评委投票表决，评委们只能表决“通过”或者“否决”（二进制嘛），“通过”票不到 2 票的软件将被淘汰。初赛之后，将结果交给懒蜗牛，由懒蜗牛决定选用哪几个软件，进入复赛。

【比赛开始】

好，现在诸位选手已经来到了硬盘里，初赛马上开始。在初赛中，每位选手首先要展示一下自己的外貌，之后下载一首固定曲目并汇报下载速度。评委根据选手的外形、速度、占用情况进行评判。有两位以上评委判选手通过即可进入复赛。哦对，忘了介绍了。本届 BT 软件大赛评委一共有 3 位，首先是 ifconfig，大家欢迎（哗哗……掌声）。ifconfig 是目前我们这里的网络专家，主要关注选手的下载能力。第 2 位是 Top，中文名叫“顶”（哗哗……掌声）。你看论坛里老发帖子说“顶”，那就是说他呢。Top 老师专注于选手的资源使用情况。第 3 位是 X，图形部门的老大，他会评判选手的外形相貌（哗哗……掌声）。当然，最终的总评委，那还是我们的用户啦。

提示：top 是查看进程状态的命令，在终端运行 top 可以看到，运行后按 q 键退出。

提示：ifconfig 是用来查看网络状态的命令。

【毒蛙 Vuze】

好，现在比赛正式开始，首先请第一位选手出场，请大家看图 4.24 所示选手的玉照。

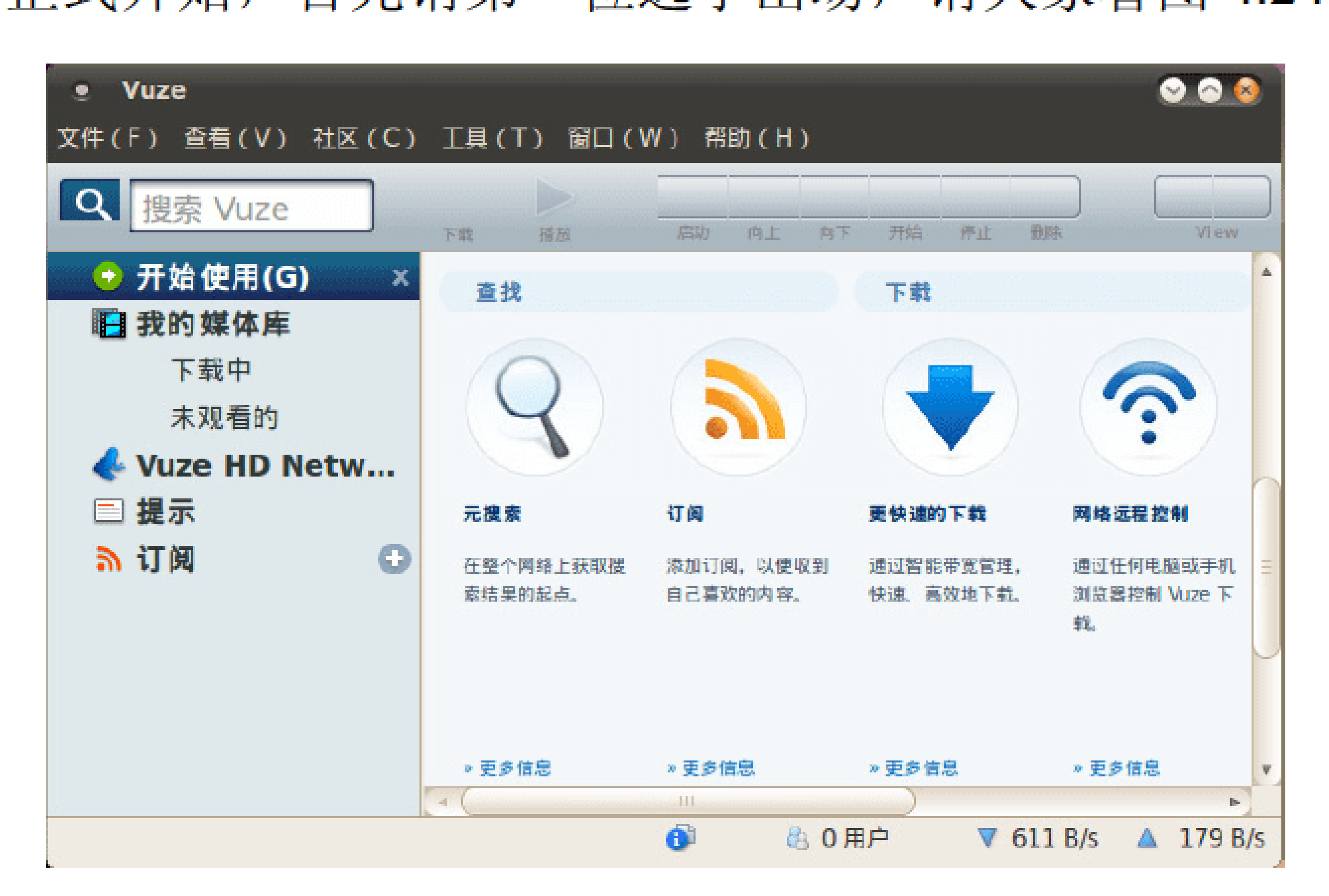


图 4.24　Vuze 界面

“大家好呱，我叫 Vuze 呱，大家可能对我的新名字不大呱熟悉，我之前呱叫做 Azureus，中文呱名字叫做毒蛙，呱呱。”“打断一下啊”，top 说道，“我看你的样子，你是个 Java 程序吧？”“恩，对呱，我是 Java 的软件。”“哦，好了，开始吧。”“好，下面我为大家演唱，哦不是，我为大家下载呱。”

他一边说着，一边拿起了固定曲目的种子文件——就是那种.torrent 的文件，然后开始了下载。X 评委看到 Vuze 选手相貌出众，仪表堂堂，下载的时候也是有条不紊，进度、速度，显示地清清楚楚，显示效果也算不错，不禁心中对他多了几分好感。这时，Vuze 已经连接到了服务器，找到了下载源，也就是找到了种子，开始下载了。不过他找到的种子不多，速度也稍微有些慢。ifconfig 评委觉得有些差强人意，不过也算合格。Top 评委则一直在下面叨咕着“漂亮倒是漂亮，这资源可也不少占啊。他这么一上台，这 200 多 MB 的内存可就没了。这 Java 的程序真是占地方”，一边叨咕一边轻轻地摇头。

等到 Vuze 下载速度稳定之后，评委叫他暂停说：“好，可以了，下一个吧。”最终 Vuze 的结果是 X，ifconfig 两评委通过，Top 否决。

【Deluge】

接下来是第 2 位选手，Deluge，玉照见图 4.25。



图 4.25　Deluge 界面

“Hello，大家好，我是 Deluge，今天很高兴能够在这里与各位老师交流，我想不管结局如何，我都会有所收获。我还比较年轻，大家可能对我的名字不如前面的 Azureus 同学那样熟悉，不过我也已经在 Windows，Linux，MacOS 都有所成就”。“哦，这些系统你都接触过？”ifconfig 问。“是的”。Deluge 回答道，“这些系统上都可以找到我的身影。”top 扭头对另外两位说：“刚才忘了问 Vuze 能不能跨平台了。不过这也不是主要的。”X 不屑地说道：“人家是 Java 的，自然能跨平台。”之后，向 Deluge 点点头说：“嗯，你可以开始了。”

只见 Deluge 拿起刚才 Vuze 的那个 torrent 文件，开始连接服务器，找种子，下载。找种子的速度比 Vuze 快了一些，下载起来速度也比较稳定。ifconfig 说道：“你看，他就比 Vuze 下载能力强，速度稳定，刚才 Vuze 是忽快忽慢，估计是因为 Vuze 找到的种子少。”Top 接着说：“嗯，身体也灵巧，才占了不到 30 MB 的空间，而且 CPU 用得也少，这个不错。”X 淡淡地说：“样子倒是中规中矩，说不上好看，倒也简洁清爽。”最终 Deluge 全票通过。

【qBittorrent】

qBittorrent 界面如图 4.26 所示。



图 4.26 qBittorrent 界面

“老师们好，我是 3 号选手 qBittorrent。”刚说一句话，只听 X 在台下小声说道：“哼，K 派的，到咱这来干活还不知道会不会整天吵架呢。”台上倒是并没有受他影响，继续说道：“我出生在法国，不远万里来到这里，把这里的 BT 事业当作是自己的 BT 事业，这是什么精神？我觉得这就是国际 BT 主义精神，这就是……”下边 Top 受不了了：“行了行了，赶紧开始吧，这段我们都背过。”

这 qBittorrent 赶紧开始，很快找到种子并且下载上了。ifconfig 一看：“这小子不错哈，这速度比那个 Deluge 又快了不少，得有 1.5 倍了。呀，又涨了，奔着两倍去了。”正说着呢，qBittorrent 一个没留神，把种子给丢了，速度瞬间到 0，重新找种子。ifconfig 擦了擦头上的汗说：“就当我什么也没说吧。”之后直接选了否决。X 却微微点头，看着旁边的 Top 说：“还可以吧？至少我看样子还可以，倒是有 K 派的风范。”Top 也表示认可：“嗯，这个占用的资源又比那个 Deluge 少了，难得难得。” qBittorrent 获得两票通过。

【KTorrent】

qBittorrent 下去后，又上来一个，长得如图 4.27 所示这个样子。他自报家门道：“偶系 Ktorrent，偶超喜爱 BT 协议，喜爱做 BT 软件，因为冷够让大家婚享快乐滴讯间，偶能够做一个酱紫滴软件，偶好高兴好高兴。”ifconfig 指着 Ktorrent 问身边的 Top：“这……这 90 后吧。”Top 无奈道：“废话，这些软件哪个不是 90 年以后的，90 年以前连 Linux 还没有呢。”X 低着头看着自己的本子向 Ktorrent 摆手：“行了行了，开始吧。”于是 Ktorrent 拿起那个 torrent 文件，开始干活，首先找种子，找了 5 分钟，没找到。ifconfig 不耐烦了：“我说……咱国庆节前还能找到不能？”Ktorrent 很谦虚：“哦，元旦以前，一定可以找到滴……”要说还是 Top 办事果断，喊：“下一个！”全票否决。

提示： BT 下载具体效果与实际实际网络环境有关，在非内网情况下，Ktorrent 并没有如此不堪。

【bittorrent】

过了一会儿，听见下一位选手说：“我呢，叫做 bittorrent，听我这名字就知道……”还没说完，3 位评委同时惊讶地抬起头，看着空荡荡的台子问道：“你在哪呢？”只听那个选手说：“我在这呢，我在这呢，你们站起来就看见了。”3 位评委站起来一看，好家伙，这哥们儿还真够小的，刚才让桌子挡上了，看不见。只听那选手继续说道：“别看我小，我可是符合 Linux 哲学之美的软件，短小精悍，我小是因为我没有图形界面，咱就专

注于干活，专注于下载。下得快，占用资源还少，这比什么都重要。”Top 听了频频点点头，直接选择了通过。X 说：“虽然这样，但是目前看用户的意思还是应该找一个有图形界面的，不是你能力不行，是这里的工作有些不适合你。”之后选择了否决。现在就看 ifconfig 了，ifconfig 向来主张实力压倒一切，所以让 bittorrent 下载一下看看速度。然而 bittorrent 也没能表现出比较优秀的速度来，还一个劲解释：“这个，是你们这里的网络的问题，你们这是内网，所以……”ifconfig 没心思听他解释，终于选择了否决。

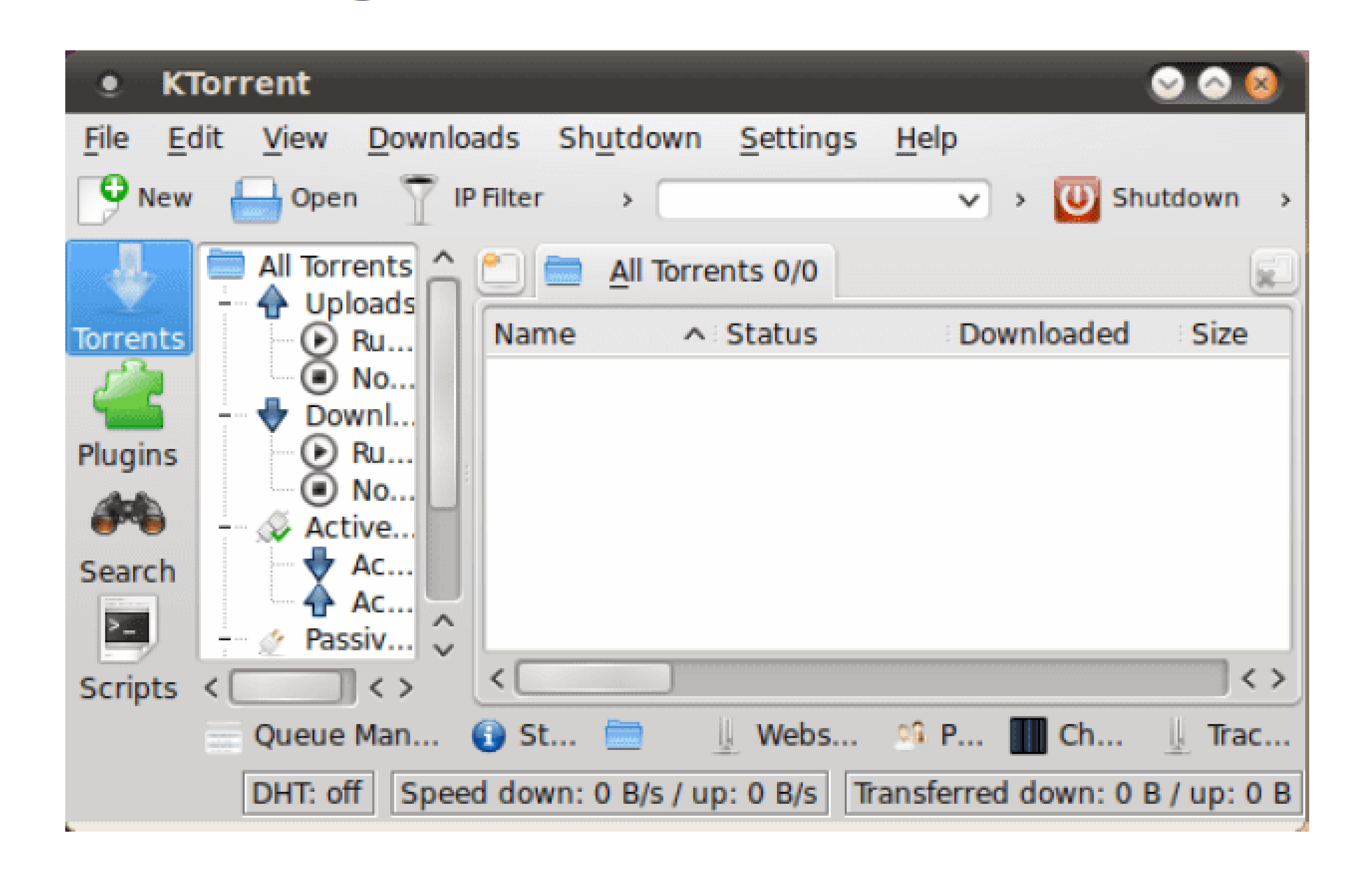


图 4.27　KTorrent 界面

【奔流】

最后一位选手还没出场，就听见他的高声朗诵：“君不见，黄河之水天上来，奔流到海不复回？君不见，高堂明镜悲白发，朝如青丝暮成雪。”念完这句，他也从后台走到了前台。只见他一派仙风傲骨，飘逸洒脱。Top 问到：“你叫什么名字？”他说：“我的名字，便在这首诗文中。”X 说：“你叫黄河？”他答道：“我叫奔流。”top 问：“听你的朗诵，你来自中国？”奔流答到：“是的。”ifconfig 说：“看看你有什么本事吧。”奔流二话不说，拿起 torrent 文件就开始忙活，很快找到了种子，下载速度也一路飙升，3 位评委都非常满意。奔流最终全票通过，不过评委们在最终的结果上做了一个标注：此软件不开源。

提示： 奔流不在软件源中，可以到这里下载：http://groups.google.com/group/benliud。

对了，忘记玉照了，见图 4.28。

初赛结果递交给懒蜗牛同学，懒蜗牛同学看了之后，决定两个获得全票通过的软件来进行复赛：deluge 和奔流。复赛的裁判只有一位，就是懒蜗牛同学自己；复赛的原则只有一条，就是用着顺手；复赛的时间，是一周。

【比赛结果】

懒蜗牛同学用了一周后，总结了一下这两位软件各自的特点。从下载的功能上来说，两位选手都不错，不过奔流依然在速度上领先一步，可能因为他生在中国，对这里的网络状况更加了解吧。

在其他周边功能上，由于 Deluge 支持插件，所以能够扩展出一些有用的功能，比如 Web User Interface 插件，可以让用户通过网络来管理这台计算机上的 Deluge 下载。如果你的电脑直接链在公网上，甚至可以在单位通过网络查看家里电脑上的 Deluge 的下载任务。

除此之外，由于 deluge 是源里的软件，能够自动跟着系统一起升级，也算是一个优点。

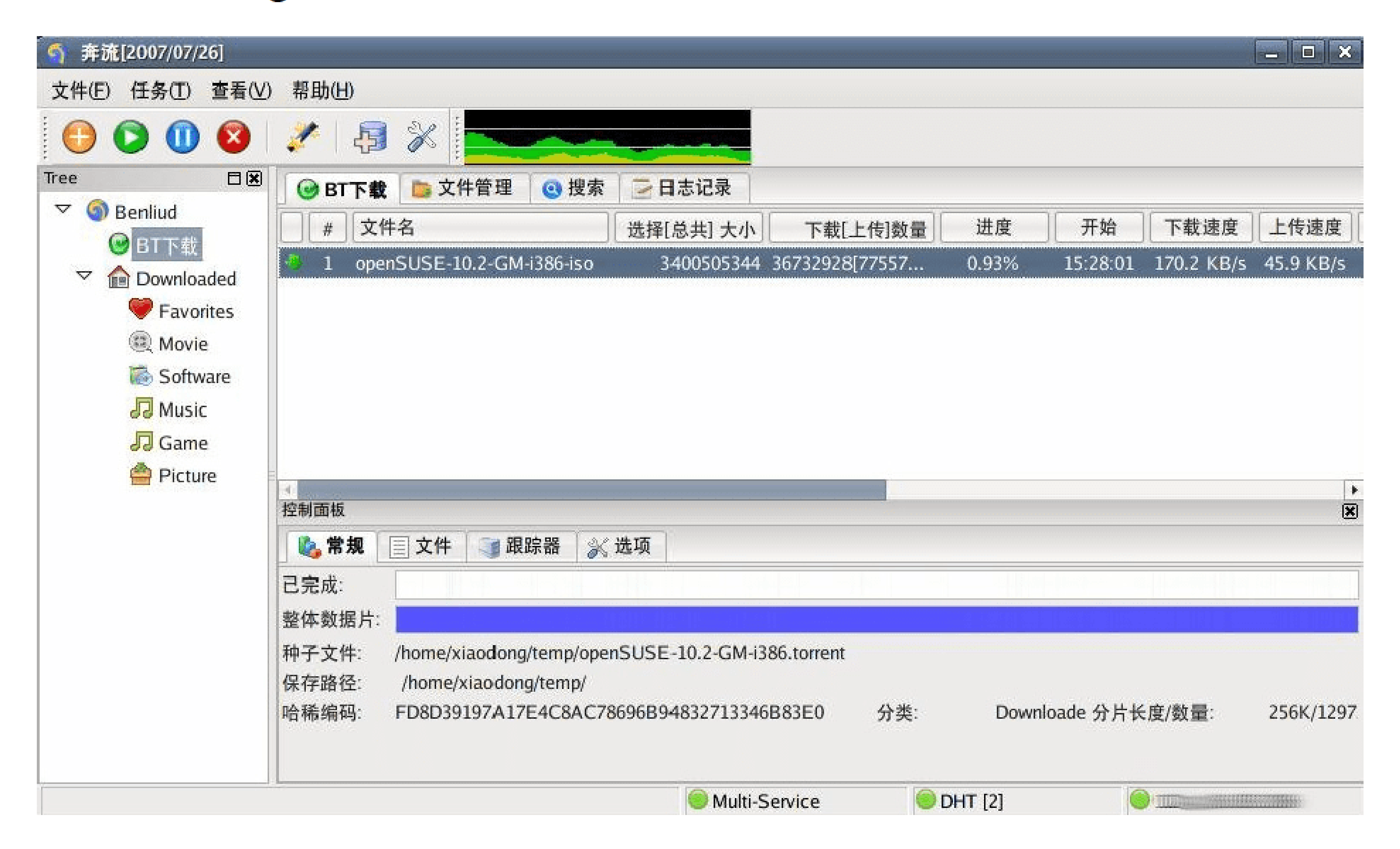


图 4.28　奔流界面

然而，懒蜗牛同学还是不大关注这些附加的功能，主张速度优先，既然奔流更快，就选择了奔流。最终胜出的奔流非常得意，下载的时候，每找到一个种子，奔流都会炫耀地大声喊“找到一个种子！”“又找到一个种子，哈哈。”当然，懒蜗牛同学也很烦，把他的这个功能关了。Deluge 呢，也没有被删除（其他的那些选手都被删除了），作为二线部队备用。

🔔**提示：**要关闭奔流找到种子时的提醒音，可以单击界面上的“设置”按钮，如图 4.29 所示。单击“设置”按钮后弹出 Preference 窗口，如图 4.30 所示。去掉“Sound tip when we got new torrent”复选框前的勾即可。

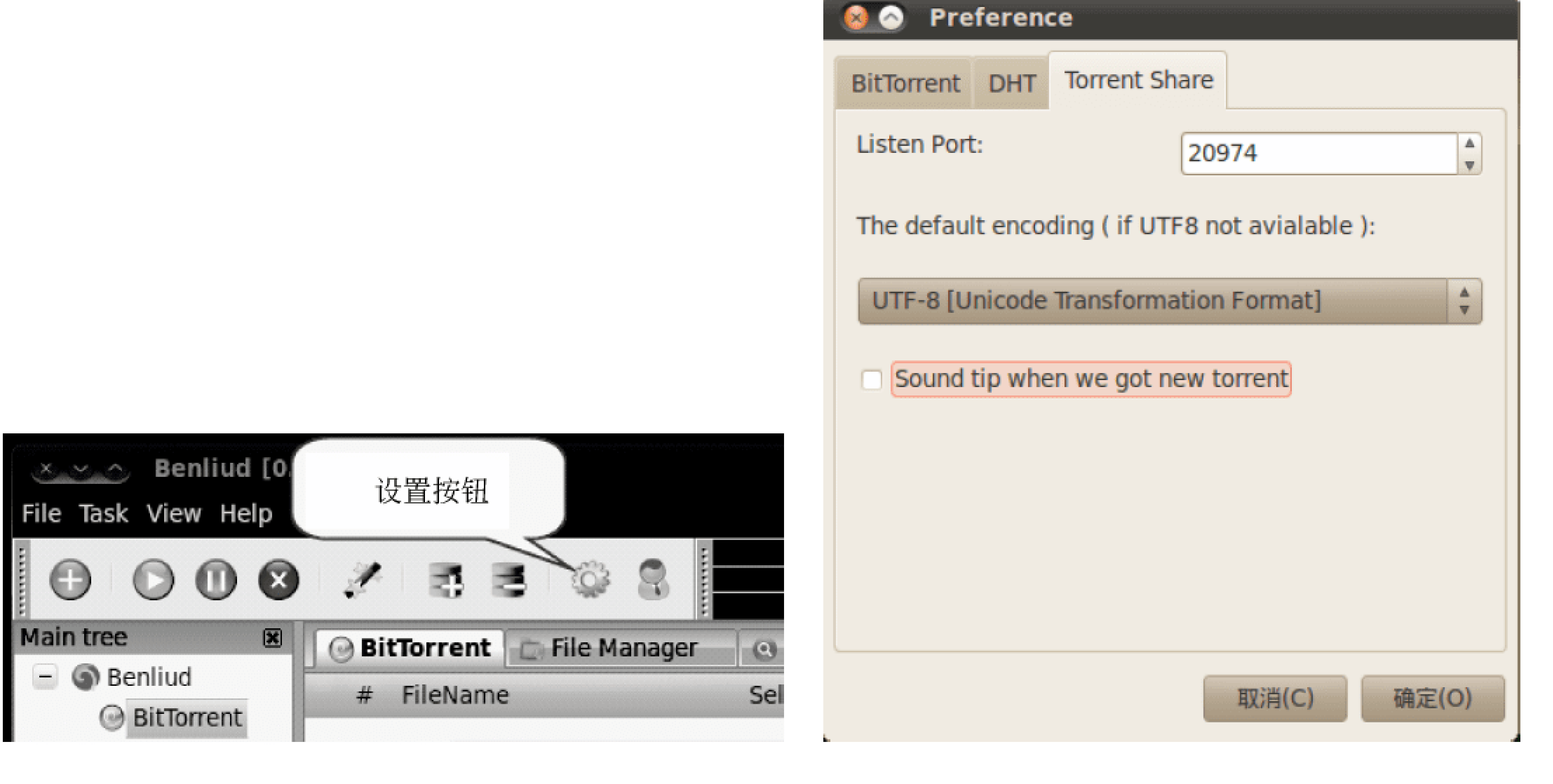


图 4.29　奔流设置按钮位置　　　　图 4.30　奔流 Preference 窗口

4.2.6　扩展阅读：软件位宽

上面提到了，我是一个 64 位的操作系统。到底这个 64 位、32 位说的是什么意思呢？

【运算位数的限制】

这个多少多少位，说的是 CPU 一次运算的二进制数字的位数。这个 CPU 就像是个计算器，我们软件用 CPU 就像人类用计算器似的。它很重要，我们要算一丁点东西，也需要用 CPU 来算（别跟我说用心算，我是软件，OK?）。

但是这个 CPU 算东西的能力，是有限制的，有什么限制呢？你拿出你家的计算器看看，算个 28+783，没问题是吧。算个 7836–473 也没问题是吧，再算个 72635446584939202937346537+1，能么？估计 99%的同志出问题了（不排除有牛人拥有很牛的计算器）：“我哪能按出这么多数来啊，我这计算器总共就能显示得下 11 位数字”。对，这就是计算器的位数限制。

CPU 也一样，一次能算的数不能无限大，总得有个边，只不过不是按照十进制的位数算的，而是按照二进制的位数算的。至于什么叫十进制，什么叫二进制，可以去问问狗狗大哥，不过不知道也没关系，咱暂时按照咱们平常的十进制来说。

【软件操作 CPU 的过程】

我们软件使用 CPU 运算的过程和你们人类使用计算器是差不多的。比如说，我这有个计算器只能算 99 以内的数字，也就是只有 2 位（也不知道谁设计的这么弱智的计算器）。那么我用这个计算机算个 3+4 怎么算呢？简单，输入 3，按+号，再输入 4，按=号，就出来了。再算个大点的，算个 56+47。先输入 56，按+号，再输入 47，按=号。咦？显示 03，怎么不显示 103 呢？废话，它倒是想显示，往哪写那 1 呀？但是我用的这个计算器（也就是 CPU 啊）是很人性化的，会提示你运算结果超出了它的能力范围。比如可能会有个红灯亮起，提示你 03 前面还有一个进位，进到百位了。

【高位宽 CPU 的优势】

好了，基本的操作说完了，现在说正题，不同位数的区别。两位的 CPU 就像刚才说的那样，那么假设现在需要计算 3173+644，这里有 2 位的 CPU 一个，4 位的 CPU 一个，分别用他们做这个计算，有什么区别呢？

咱先看这 2 位的，有人说了，2 位的只能算两位啊，这个没法算哪？唉，这机器是死的，咱软件是活的啊，一次只能算 2 位，咱不会分开多算几次么。首先，输入 73，按+号，再输入 44，按=号。显示出来 17，同时红灯一亮，说明还得进位。好，找张纸记下 17 这个数，还得写上“得进位”。然后再输入 31，按+号，输入 6，按=号，显示出来 37。别忙，没完，刚才还得进位么不是，再输入 37，按+号，输入 1，按钮，咔嚓，出来 38。好，最后结果拼一块，高位是 38，低位是 17，最后结果：3817。

再拿这 4 位的算算看。4 位的就意味着输入的和显示的数最大可以是 9999，也就是说我直接就可以输入 3173，按+号，再输入 644，按=号，显示出来 3817，OK，收工。

🔔**提示：**目前市场上常见的 CPU 都已经是 64 位 CPU。

【软件位宽与 CPU 位宽的关系】

这就是 2 位的 CPU 和 4 位的 CPU 的不同，从理论上来说，4 位的要比 2 位的快，从上面的例子看得很明显嘛，大一点的数，4 位的 CPU 一下就能算完，2 位的 CPU 要折腾好几次。但是这 4 位的 CPU 还得有人会用才行，这就需要 4 位的软件来用这个 4 位的 CPU。

终于说到软件的位数了，CPU 的位数就是一次能计算多少位的数，那软件的位数呢？就是说明这个软件需要使用多少位的 CPU。软件干活肯定需要计算，计算就得用 CPU，2 位的软件会用 2 位的 CPU，4 位的软件就会用 4 位的 CPU（还是拿十进制位做比喻啊）。比如有一个 2 位的软件，当他运行在一个 2 位 CPU 的电脑上的时候就是这样：比如要算 3173+644，他就会先算 73+44，然后记住进位，然后计算 31+6，然后加上进位，最后拼起来，得到答案，就像上面描述的那样。那么当这个 2 位的软件运行在一个 4 位的 CPU 上的时候会怎么样呢？他会先算 73+44，然后记住进位，然后计算 31+6，然后加上进位，最后拼起来，得到答案……有人说了，他怎么不直接算啊？4 位的 CPU 不是能直接就算出来么？但是别忘了他是 2 位的软件啊，他不会用 4 位的 CPU，但是不会用不等于完全不能用，他还是可以拿 4 位的 CPU 当成 2 位的来用，只是有些浪费而已。

提示：64 位 Ubuntu 系统中自带的软件，以及通过软件源安装的软件，除某些不开源的软件外，都是 64 位的。

那么要想完全发挥 4 位 CPU 的性能该怎么办呢？当然就得 4 位的软件出场了。当一个 4 位的软件运行在一个 4 位的 CPU 上时怎么计算 3173+644 呢？大家大概都知道了，直接算，一次完成。那么当一个 4 位的软件运行在一个 2 位的 CPU 上时会怎么样呢？这个软件会写个 3173 的纸条要往 CPU 的寄存器里塞，急得满头大汗就是塞不进去，最后一甩手——老子不干了，这破 CPU 没法用！当然，这只是个比喻，并不是说 4 位软件在 2 位 CPU 上算 3173+644 就算不了，算 1+1 就能算。4 位的软件是根本无法运行在 2 位的 CPU 上的。

4.2.7 扩展阅读：进程

咱说 Chrome 是个多进程的浏览器，一运行就复制出好多进程来。有人可能对进程这个名字还不是很明白，什么是进程呢？

【进程的概念】

简单地说，进程就是正在干活的软件。比如狐狸妹妹，躺在硬盘里睡觉的时候她就是一个软件、一堆数据、一坨代码。当她被叫醒，跑进内存里开始干活的时候，她就是一个进程了（当然，其实这么说不很准确，但可以姑且这么理解）。换句话说，内存里忙忙碌碌的，都是一个个的进程。当然，同时他们都是程序、都是软件，这不冲突。

就像去公司上班的人，他们都是人，当他们在公司工作的时候，他们都是公司的员工。员工，就像进程一样。很多公司的员工每个人都有个工号，什么 NB001，SB999 之类的；每个进程也都有一个唯一的标识——进程 ID 号，简称 PID。这个 ID 号是由我分配给每一个跑进工作间的进程的，分配的规则很简单，每人一个，每次加一。第一个跑进来的就是 1 号，在我们 Linux 系统里，有个叫 init 的家伙每次都是第一个被我叫起来，帮我打理一下日常工作，所以他的 ID 号总是 1。而且，他还有个特殊身份，这个咱暂时保密，待会儿再说。

🔔提示：init 一般位于/sbin/目录下，内核启动后会首先调用此程序，进行一些初始化工作。

【管上级叫爹】

每个公司的员工都有个直属的上级，上级又有上级，依此类推。我们这里的进程也是这样的，只不过我们不叫“上级”或者“上司”，我们叫——爹！好吧，似乎这个称谓土了点，但是就是这个意思。一个进程之所以成为一个进程，一定是由于另一个进程创建了他（有点绕嘴吧）。比如说用户开了一个终端，于是就有了一个 bash 进程，然后用户在这个终端里敲入 firefox 并回车，bash 就去找狐狸妹妹，把她带到内存里开始工作，于是就创建了一个 firefox 进程。好了，现在，firefox 这个进程是由 bash 这个进程创建的，那么，bash 这个进程就是 firefox 这个进程的父进程，firefox 进程就是 bash 进程的子进程。也就是说，狐狸妹妹就得管 bash 叫爹！那 bash 也得有个“爹”吧？是的，如果是在 Gnome 环境下开的那个终端，那么 bash 他爹就是调用 bash 的 gnome-terminal。

既然每个进程都有爹，爹进程又有爹，如此循环往复，肯定有一个站在金字塔最高点的总“爹”吧？难道，难道笨兔兔你就是他们的总爹？很遗憾，我不是，所有进程的总爹，是每次启动第一个被我叫起来的 init。所有的进程都是被 init 直接或者间接创建的，所以 init 才是所有进程的祖宗！

关于父进程，有两点要说明。

第一，我们这的父子关系不是固定的，是会变换的。如果从 bash 启动 Firefox 那么 bash 就是 Firefox 的爹，如果直接从图形界面启动那就没 bash 什么事情了（这时候 Firefox 的爹其实是 init）。

第二，不要问我哪里有妈进程！

🔔提示：父进程的说法源自英文 parent process。

【当爹的义务】

当爹也有当爹的义务，人家不能白叫你一声爹是不是。当自己的娃（也就是子进程啦）做完自己该做的工作以后，就停止了一切动作，像个死尸一样待在那里，当爹的就负责给他“收尸”。

一个结束了所有工作的进程，会处于一种“僵尸”状态，这时候他什么也不做了，就等着被干掉。进程进入僵尸状态前一般会通知他爹一声，汇报一下说：爹啊，俺已经把该做的都做啦，现在我要变僵尸啦！（让后平伸双手开始行走？那是生化危机！）然后他爹负责向我汇报：我家娃干完活了，你把他的工号（就是 PID，记得吧）取消掉然后让他回去睡觉吧。然后我就把它的工号收回，看看他有没有什么申请了没释放的资源（一般一个好孩子在结束运行成为僵尸之前，会主动释放掉自己申请的资源的）。确认都没问题了之后，他就被从我的进程列表中清除了。

🔔提示：所谓进程申请的资源，包括进程申请过的内存、打开的文件、Socket 连接等。

【当爹遇到意外】

但是有时候也会有些特殊情况，比如有的时候娃还在兢兢业业地干活呢，结果他爹死了（可能他爹干完活退出了，也可能被用户用命令 kill 了）。这个时候我就会发个信号给他家娃说：那个……娃呀，那啥，跟你说个事，你爹死了。

这时候有的娃就悲痛欲绝：俺爹都死了俺活着还有啥意思啊，呜呜呜……俺也僵尸吧。然后他就退出了。比如你在终端运行 Firefox，然后把终端关了，Firefox 也就退出了。

也有的娃比较坚强，一定要完成上级交给的任务，化悲痛为力量。这时候我会给他找个新爹——因为每个进程总得有个父进程，没爹是不行的。一般我会安排他爹的爹来当他的爹（又绕进去了吧），也就是这个进程原来的“爷爷”进程来当他的父进程。然后这娃在长了一辈后，继续认真工作。比如你在终端运行 nohup firefox，然后把终端关了，Firefox 继续运行。那如果他爷爷不幸也挂了呢？那就继续往上找吧，我们说了 init 是所有进程的祖宗，所以他那里就成了最终的“无依靠青年进程收容所”。

如果 init 也挂了呢？那系统就挂了，重启吧！

提示：父进程退出时，系统会向子进程发送“挂断”信号（SIGHUP），子进程是否退出取决于子进程如何处理挂断信号。

【当爹和娃同时出意外】

还有的时候娃已经把该做的事情做完了，汇报给他爹并变成僵尸。可是他爹还没来得及给自己娃收尸，自己就先挂掉了，这个时候就有点麻烦了。

首先我没法通知那娃说她爹挂了，因为那娃已经是僵尸了，啥也不听啥也不干了。其次我也不能直接把他干掉，啥事情都得按规矩来嘛，只有他爹向我申请我才能把他干掉，可是他爹又已经挂了……那怎么办呢？那就按流程来，先给这个娃找个爹，哪怕这娃已经是僵尸了，也得有个爹。一般我会找到 init 说：那个 ID 号是 2725 的进程爹死了，你当他爹吧。一边说一边看也不看地用手往那边一指，假装自己没看到那娃已经成僵尸了。一般 init 也不会太注意，直接就答应了，然后马上发现了事情的真相，跑到我这里来说：那娃已经成了僵尸啦，你还叫我收养个啥？我肯定会一脸无辜状：啊？是啊，那不管怎样，你是他爹了，你负责处理一下后事吧。于是 init 只好以爹的身份处理那个僵尸的后事，问题就这样解决了。

4.3 我的影音生活

狐狸妹妹今天接到懒蜗牛同学的任务，要去下载一部叫做《Big Buck Bunny》的电影。说是电影，其实就是个短片，还不到 10 分钟。并且它还有个最大的特点——它是开源的。好，废话不多说，狐狸妹妹已经用她的 Downthemall 扩展把这个短片下载下来了，于是，就引出了一场播放器之间的斗争。

提示：所谓开源电影，是说它是在开源的平台上用开源的软件制作的，并且免费下载观看，还可以获得它的原始制作文件。

4.3.1 简约的 Mplayer

下载下来之后，懒蜗牛同学就找来播放器播放这个短片。虽然之前懒蜗牛同学安装了 Gnome Mplayer 播放器，但他还是觉得 Gnome Mplayer 的界面简陋，而且看片的时候其实

基本不需要怎么操作，那还不如干脆不要这界面，直接让 Mplayer 来播放。那样就光一个视频窗口，其他的什么按钮都没有的界面才叫简约，而且也很拉风，因为在 Windows 下没有这么看视频的。

想到这里，说干就干。懒蜗牛同学找到了下载好的那个.ogg 文件，.ogg 是一种视频文件格式，Mplayer 不用安装额外的解码器就可以播放。懒蜗牛右击这个文件，选择“属性”，单击“打开方式”标签，如图 4.31 所示。

打开方式标签里面显示了目前在我这里注册过的，所有可以用来打开这类文件的软件。包括 Gnome Mplayer、电影播放机、Rhythmbox 等。并且电影播放机是默认程序，也就是双击一个.ogg 文件，就会用电影播放机打开——而这些都不是懒蜗牛想要的。

懒蜗牛想用 Mplayer 来打开这个文件，但是这里没有，于是他单击了“添加”按钮。然后我就给他列出了一个软件列表，都是已经安装了的软件，想用哪个软件打开这个文件，在这里选就是了，如图 4.32 所示。



图 4.31　文件属性打开方式标签　　　　图 4.32　选择用以打开文件的程序

不过这里列出的都是有图形界面的软件，Mplayer 并不在其中。但是没事，懒蜗牛很熟练地单击了下面的那个“使用自定义命令”前的“+”号，然后在弹出的文本框里输入了 Mplayer，就像图 4.33 这样，最后单击“添加”按钮。

这样，就回到了“打开方式”那个标签，上面已经多出了一个没有图标的 Mplayer，但还不是默认的，再把他选成默认的程序就可以了，就像图 4.34 这样。之后直接关闭就可以了。这样做之后，再双击那个刚刚下载的 OGG 文件，Mplayer 就自动跑出来播放了。只见屏幕上只有一个简约得不能再简约的窗口，窗口中就是正在播放的视频——一只可爱的大兔子。

提示： Gnome 的设置界面多数没有确认按钮，设置好后即生效。



图 4.33　手动输入用于打开文件的程序

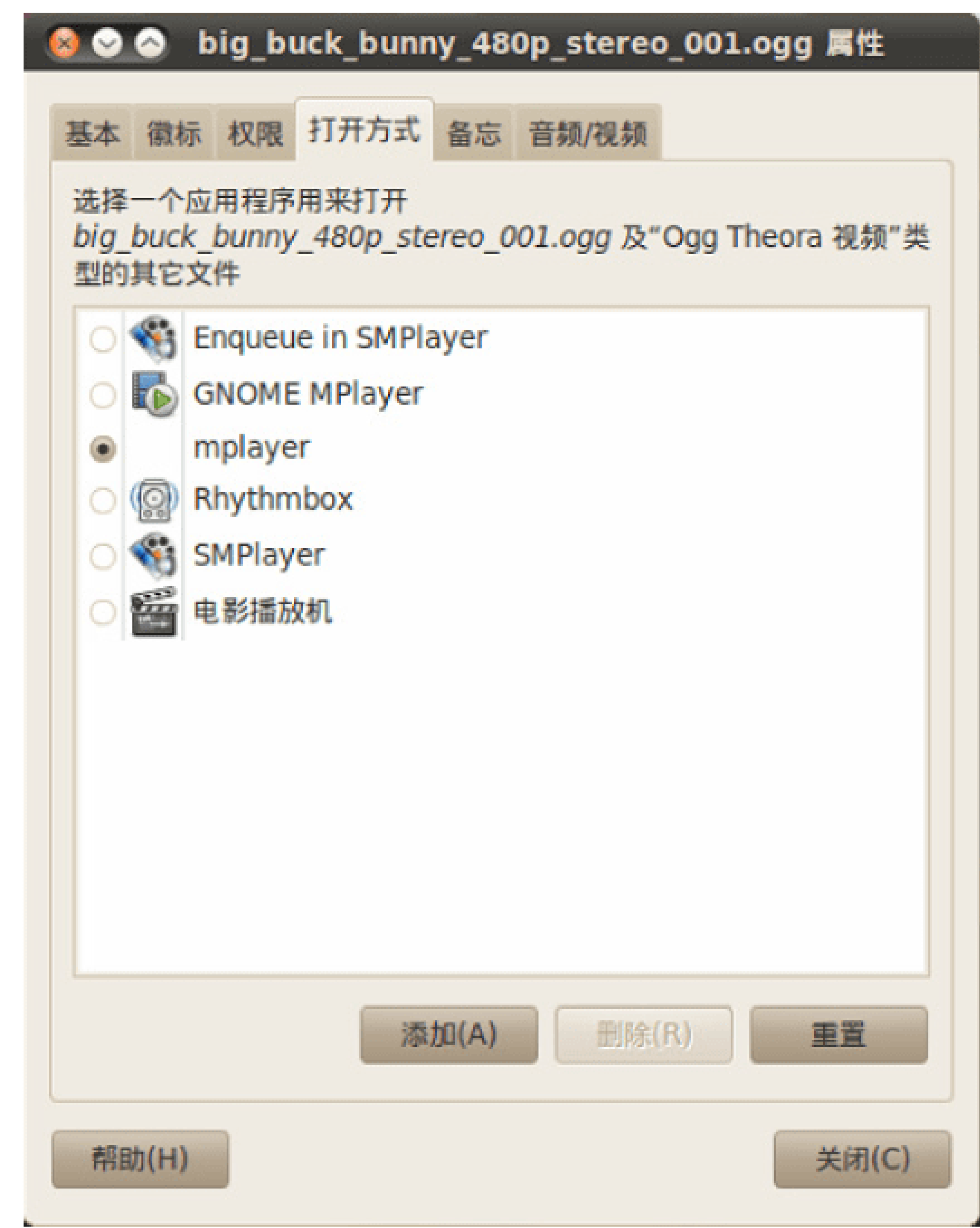


图 4.34　添加 mplayer 作为默认打开方式

有人会说，这界面上啥也没有，我想暂停、快进、调整音量啥的怎么办？不用担心，Mplayer 早为你想好了，整个键盘就是你的操作界面。所有操作都有快捷键，常用的如表 4.1 所示。

表 4.1　Mplayer常用快捷键

按　　键	说　　明
p	暂停（暂停后按任意键播放）
方向右	快进 10 秒
方向左	后退 10 秒
方向上	快进 1 分钟
方向下	后退 1 分钟
PageUp	快进 10 分钟
PageDown	后退 10 分钟
o	显示当前视频进度
0	提高音量
9	降低音量
+	增加音频延时（影音不同步时使用）
-	减少音频延时（影音不同步时使用）
1	降低对比度
2	提高对比度
3	降低亮度
4	提高亮度
f	全屏/退出全屏

懒蜗牛同学早已熟悉了这些快捷键，现在操作起来得心应手。

4.3.2　强大的 SMplayer

作为一个喜欢简约的人，懒蜗牛同学觉得自己能操作一个这么精简的播放器，很有成就感。不过，不是所有人都受得了只有一个窗口的播放器的。

【从懒蜗牛同学和 MM 的对话开始】

这一天，一个陌生的，温柔的手指触摸到了我们这台电脑的键盘，接着，一个清新的声音透过一直插在电脑上的麦克风，回荡在紧张忙碌的工作间里："呀，你的电脑真好看，你这是 Windows 7 吧？"哎，一听就是个外行，Windows 7 哪有我漂亮，哼。随即听懒蜗牛说道："这个不是 Windows，是 Linux。"工作间里的同志们会心地点点头。

"哦？Linux？是微软新出的吗？"哦，除了微软不认识别的公司了啊。

"不是，这个跟微软没关系，Linux 是一个自由的操作系统，免费的哦。"

"免费的？那能好用么？"姐姐，便宜也有好货的。

"你试试啊，挺好用的，至少界面就好看。"

"那倒是，嘻嘻。"

"其实你可以装一个看看，反正你不怎么玩游戏。看个片，上个网什么的很方便的。"

"对了，你这里有什么有意思的电影么？"

"有啊，有个动画片挺有意思的。"懒蜗牛说着，双击了那个.ogg 文件，简洁的 Mplayer 出现了。由于省去了图形前端，Mplayer 启动很迅速，光秃秃的样子也很让那个 MM 惊讶。

"这是什么播放器啊？"

"Linux 下的最好用的播放器，怎么样，简洁吧？"懒蜗牛同学很得意。

"真难看。"得，碰钉子上了。

"这样什么操作界面都没有，难道每次必须从头看？也不能停？"MM 继续说。

"当然不是"，懒蜗牛同学赶紧解释："都可以用键盘操作的，你看这个是播放，这个是快进，这个调节音量……"懒蜗牛同学一边解释，一边在键盘上演示着。

"还要记住那么多键啊，好麻烦。"我承认，艺术本来也不是每个人都欣赏得了的。

"也有普通的图形界面的播放器，比如这个……"懒蜗牛关掉了光秃秃的 Mplayer，打开了 Totem。

"嗯，这个还顺眼点，不过好像功能也不强大，有没有暴风影音？"

"要暴风影音那样的播放器也有啊，等我装一个你看看。"说着懒蜗牛打开了新立得，查找 SMplayer 并安装。数分钟之后就装好了，超级牛力干活没得说。

"这就装好了？怎么没见到你上网下载啊？"

"这个……嗯……这就是 Linux！"懒蜗牛同学，我理解你，要给这家伙讲明白是有点难。

"哦，这还挺方便的。咦，这个界面好多了嘛，你之前怎么不用这个？"

"我喜欢那个精简，速度快。Linux 的软件就是这样，什么样子的都有，总有一款适合你，哈哈。"

说着笑着，那个大兔子片就结束了，懒蜗牛又让 SMplayer 去播放一个叫做"见过大爷.rmvb"的片子，并且和 MM 安静地看起来。SMplayer 其实跟 Gnome Mplayer 一样，都

只是图形前端，还是离不开 Mplayer。但是 SMplayer 的厉害之处在于他能够充分发挥 Mplayer 的功能，提供更多可配置的选项，并且操作起来也更符合多数用户的习惯。可以说，SMplayer 和 Mplayer 合作起来，那真是珠联璧合，黄金搭档。我们来看一下 SMplayer 的英姿，如图 4.35 所示。

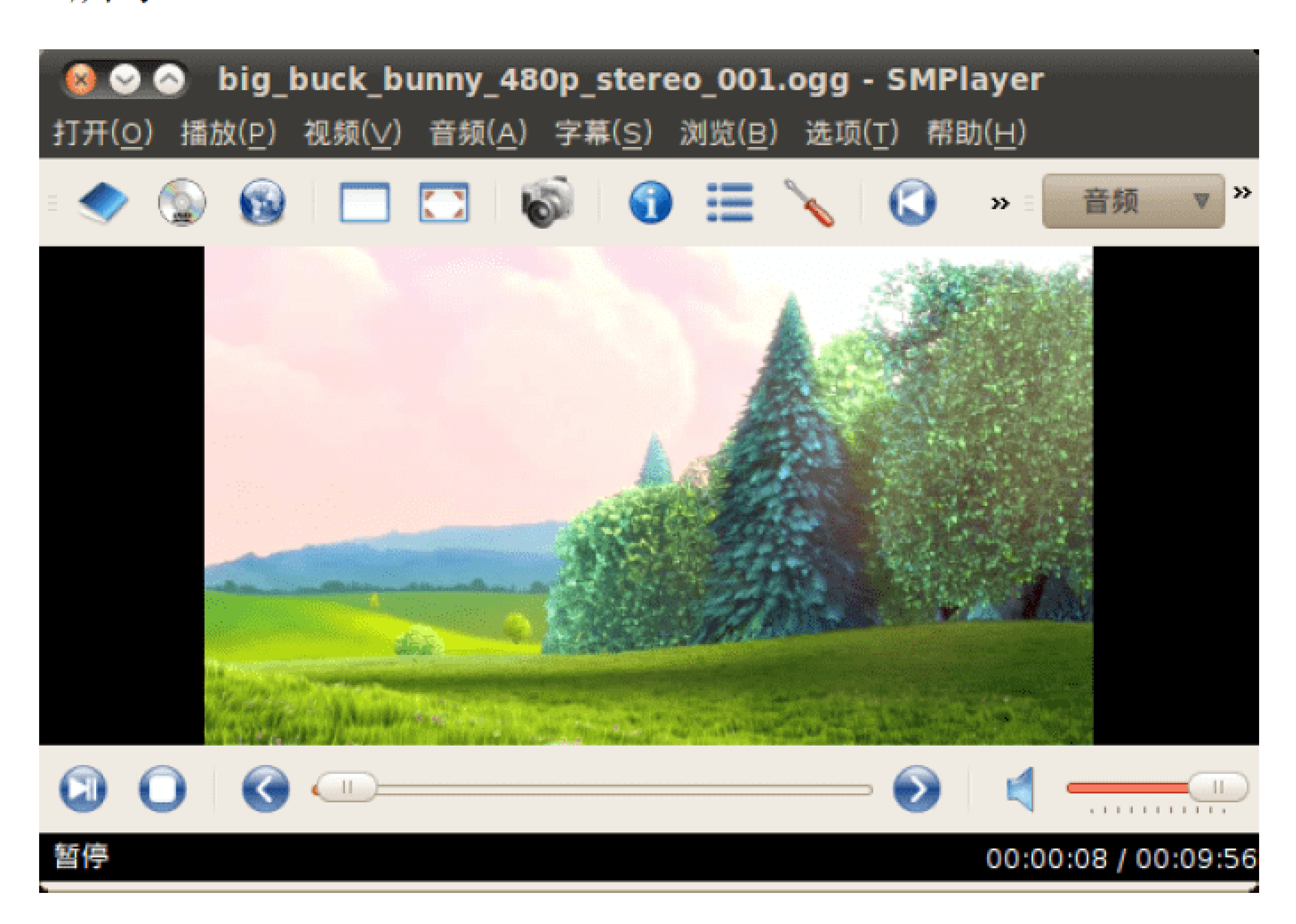


图 4.35　SMplayer 外观

提示：在 SMplayer 和 Gnome Mplayer 下，Mplayer 的快捷键依然有效。

【江湖上 G 帮 K 派各占半边】

懒蜗牛和 MM 看着片子倒是安静了，殊不知工作间里面这会儿可闹腾起来了……

刚才懒蜗牛用 Mplayer 播放视频，后来换成了 Totem，最后又改成了 SMplayer。这么一折腾，Mplayer 老成持重倒是没什么，那被替换下来的 Totem 显然很不服气，抱怨说："要论真本事，我也未必比谁差，不就是长得漂亮点么，哼！" 这 SMPlayer 听不过去了，一边忙着播放视频，一边反驳："你本领不差，难道说我本领差？你能记住每个视频上次播放到什么位置么？我可不光是靠长得漂亮。不过话说回来，我们 Qt 的程序，确实比你们 gtk 的细致了不少，用户爱看我们，这也是没办法的事情。" Totem 怒目而视："你别忘了，现在内存里面可是我们 Gnome 环境，你敢说这种话，有点欺人太甚吧！" "Gnome 怎么啦，Gnome 就是不如 KDE 好看，有错么？" 我一听，坏了，这俩又要打起来。

提示：SMplayer 可以记录视频播放的位置，如果播放到一半时关掉视频，下次再打开同一视频后，直接从上次关闭的位置开始播放。

话说在 Linux 这片恩怨情仇的大地上，做图形界面这行当的，以两大帮派为主。一个是 Gnome 帮，帮中的软件们都修炼 gtk+宝典，帮众们信奉简单高效的做软件准则。什么事情都力求用简单的界面来实现，并不留给用户太多可以设置的东西。G 帮认为，对于初级用户，不要搞那么多设置项，搞得用户头昏脑胀。能默认的都默认好了，打开软件就工作。而对于需要自定义的高级用户来说，直接去改配置文件就好了，这难不倒高级用户。

提示：gtk+是一个用于创建图形用户界面的程序库。提供 C 语言风格的函数接口。Gnome 就是基于 gtk+开发的。

另一大帮派，就是 KDE 派了，K 派们都修炼 Qt 大法，派中的软件都觉得界面要做得方便，易用，易于配置，坚信细节决定成败。界面要细腻，要漂亮，让人家一看就喜欢，这才是好的图形界面软件。所以 K 派的软件都有好多可配置的选项，新手可以无视，老手配置起来很方便。两帮派观念不同，本来也没有谁强谁弱，但是偏偏有时候还是会争个风头，动不动就吵个天翻地覆。本来我这里都是 G 帮的软件，结果今天懒蜗牛偏偏安装了 K 派的 SMplayer，这不就吵起来了。

提示：Qt 也是一个用于创建图形用户界面的程序库。提供 C++语言风格的函数接口。KDE 就是基于 Qt 开发的。

【小图腾心不甘挑起事端】

Totem，那可是 Gnome 的嫡系软件，自带的播放器。本来被用户换下来，他心里就不舒服，再一听 SMPlayer 这么说他，更是无名火起，顺手抄起一把兵刃——也就是一个视频文件，指着 SMplayer 说：“有本事，别光耍嘴上的功夫，咱们过两招瞧瞧。”不等 SMplayer 答话，唰唰唰原地耍了起来，只见他一招一式，无甚惊人之处，却招招使得灵活熟练，那视频文件就好像本来写进他自身的代码段里一般，什么播放、暂停、前进、倒退、全屏、截图、调横纵比、显示字幕，虽说只是播放器的粗浅功夫，难得的是样样做得恰到好处，不温不火。

这时那 SMPlayer 正在给懒蜗牛同学和 MM 播放视频呢，本不该分神，却忍不住争强好胜之心，也找个兵器练起来。他一边给懒蜗牛播放，一边跟 Totem 比武，双手同时耍开两个视频，速度自然慢了一步，但却是花样迭出。什么音画时差调整，什么字幕控制，网上查找字幕，降噪，反拉丝，旋转屏幕，专门找 Totem 不会的招式操练。Totem 见对方同时耍着两个兵器，虽说稍稍缓慢一些，却依然稳扎稳打，何况招式确实比自己多，自觉是落了下风。

可 Totem 又怎肯示弱，扔下手上的兵刃，又从兵器架上抄起另一把，练了一会儿又换一把，顷刻间，已经换了 18 件兵刃。什么 RMVB、MOV、AVI、WMV、FLV、CD 音轨、MP3、MP4 等，件件拿得起，放得下。那意思，别看我会的招式不如你多，但我能耍的兵器可不少。SMPlayer 见此景微微一笑，扔了手中的家伙，把 Totem 换下的兵器一件件拿起来，依次也耍了一通，也是样样精通。只看得 Totem 一身冷汗，自知自己本领也就到此为止了，可是却如何下得了这个台？

正自思虑之间，只听得身背后有人言道：“贤弟，你且下去歇息，哥哥我来会儿他！”

【同门兄弟间又起波澜】

只见 Totem 身后闪出一人，正是 Gnome Mplayer。一看这名字您就知道了，这是 G 帮的人。SMplayer 一看见他，顿时心里犯嘀咕。为什么呢？咱说过，这 Gnome Mplayer 和 SMplayer 其实都不过是个图形界面的外壳而已，他们的后台都是 Mplayer 老大。因此，这二人本是同门兄弟，只因信仰不同，故而一个入 G 帮，一个投 K 派。既是同门学艺的兄弟，本领又能差几何？

提示：Totem 播放器不是 Mplayer 的图形前端，不依赖于 Mplayer。

这 SMplayer 寻思，纵然自己比 Gnome Player 勤学苦练，无奈方才刚刚跟 Totem 大战一场，又要照顾着给用户播放的视频，自己恐难取胜。可那 Gnome Player 更无废话，上来就抄起兵刃比划起来，也是将一件件兵刃轮番耍起来，比刚才 Totem 所耍的还要多上一些。

SMplayer 只得咬咬牙，再跟 Gnome Player 拼死一战。霎时间整个内存当中，那真是录像与配音齐飞，字幕共长片一色。二人斗得难解难分。

这 Gnome Mplayer 平时便不服 SMplayer，本是同门学艺，可在江湖上，绿林中，这 SMplayer 的名号却远远盖过他 Gnome Mplayer。今天他就想让别人见识见识，他 Gnome Mplayer 不比 SMplayer 差。因此他论起武艺，虽然与 SMplayer 差着一点，但是招招都拼尽全力，狠辣异常。斗到 130 多回合的时候，这 SMplayer 渐渐支持不住，眼看就要跳帧了。

【老先生劝罢斗各回各边】

这时候，内存里已然围满了人。有看热闹的，有叫好助威的，有打酱油路过的，还有不明真相的群众，围得里三层外三层。因为现在的图形环境是 Gnome，因此这内存里围观的基本都是 G 帮的人，SMplayer 可谓孤军奋战了。

正在这时，OpenOffice 老先生推推眼镜，咳嗽了一声，发话道："两位请住手！"OO（以下特指 OpenOffice）老先生可谓德高望重，虽然本身是 G 帮的人，可基本上所有的 Linux 发行版，不管是用 GNOME 也好，用 KDE 也罢，都会请他做默认的办公软件。因此即便是 K 派的人，也都服这位老先生。话音刚落，GNOME Mplayer 罢手不打，SMplayer 自然也愿意停下来休息，站在那里直喘粗气。只听 OO 老先生继续说道："要我说啊，这机器启动时进入的是 GNOME，满内存都是 G 帮的人，可用户偏偏要叫人家 SMPlayer 来播放视频，这已然是人家赢了。更何况两位轮番上阵，不是也没把人家一人打下来么？"后面这句，自然是看着 Totem 和 GNOME Mplayer 说的。Totem 艺不如人，自觉惭愧。GNOME Mplayer 愤愤不平，却也无话可说。一场风波暂时算是平息了。

提示：KDE 环境中有 Koffice 办公软件，但由于兼容性和稳定性的问题，即使是基于 KDE 环境的发行版，也都使用 OpenOffice 作为默认的办公软件。

4.3.3 琳琅满目的音频播放器

内存里播放器们折腾的时候，懒蜗牛同学和 MM 一边看着片子一边闲聊，倒是十分惬意。他们聊到生活，聊到感悟，聊到春花秋月，聊到诗词歌赋，还聊到一首歌，据 MM 说是感人至深，发人深省，于是 MM 走后，意犹未尽的懒蜗牛决定去找来听听，聊以慰藉今日畅谈。

播放音乐这件事情，基本上 Mplayer、Totem 等视频播放器都可以完成，因为本来一个视频中就包含视频编码和音频编码，除非你看的是哑剧。但虽然如此，人们一般还是会需要一个专门播放音频的软件。

视频软件专注于怎么将视频解析得更加清晰，播放得更加流畅。音频软件要解的码量少，不是专业人士，对解出的音频质量也没有更多的要求，而更重要的是对文件的管理，和用户的交流。音频播放器就像一个图书馆的管理员一样，要对自己的资源了如指掌。

【系统自带的 Rhythmbox】

我们这里，系统刚刚装好的时候就有专业的音乐管理员，就是图 4.36 所示这位，已经跟大家见过面的八音盒——Rhythmbox。这哥们儿长相一般般，不过功能不差，一般的音乐自然都能摆平，网络电台啥的也没有问题，还能获取歌词和专辑封面（不过基本都是英文歌曲）。管理起音乐文件来初看似乎有些凌乱，不过熟悉了之后就明白他管理得简单高效。并且因为是 Gnome 的默认程序，他的全局热键也让很多人趋之若鹜。



图 4.36　Rhythmbox 界面

【熟悉的倩影——Audacious】

离家千里，难免思乡。手里的东西用得熟络了，也不免难以割舍，软件大约也是如此吧。有的用户在 Windows 7 那里识得一个叫做芊芊的音频软件，简洁的界面和强大的功能是她吸引人的地方。好在我们这自由的世界里，也有这么一位类似的佳人，虽然论功能差了一些，但也足够认识芊芊的人追忆往昔，她就是 Audacious，图 4.37 里这位。认识芊芊的人，恐怕一看到 Audacious 的样子，便可知何为伊人倩影了。



图 4.37　Audacious 界面

【非常 OK 的 Amarok】

若说功能和外观，音频软件里，当属 Amarok 了，看图 4.38 就能看出他的强大。Rhythmbox 的那些功能 Amarok 自然都有。并且，Amarok 英姿飒爽，不免更加惹人喜爱。于是有国人为他开发了插件，其中，便有下载歌词和封面的，这回自然是中文的了。只不过他身为一个 K 派，在 Gnome 环境下似乎总是难免有些碍手碍脚，不得施展。

【Gnome 下的 Amarok——Exaile】

Gnome 的用户也别急，G 帮同样不乏精英，那就是有“G 帮的 Amarok”之称的 Exaile，就是图 4.39 里这位，是不是跟 Amarok 有些类似之处？Exaile 追求着 Amarok 的追求，感悟着 Amarok 的感悟，同样的精美，同样的能力，让他们虽不在同一个门派，却有着同一个梦想。

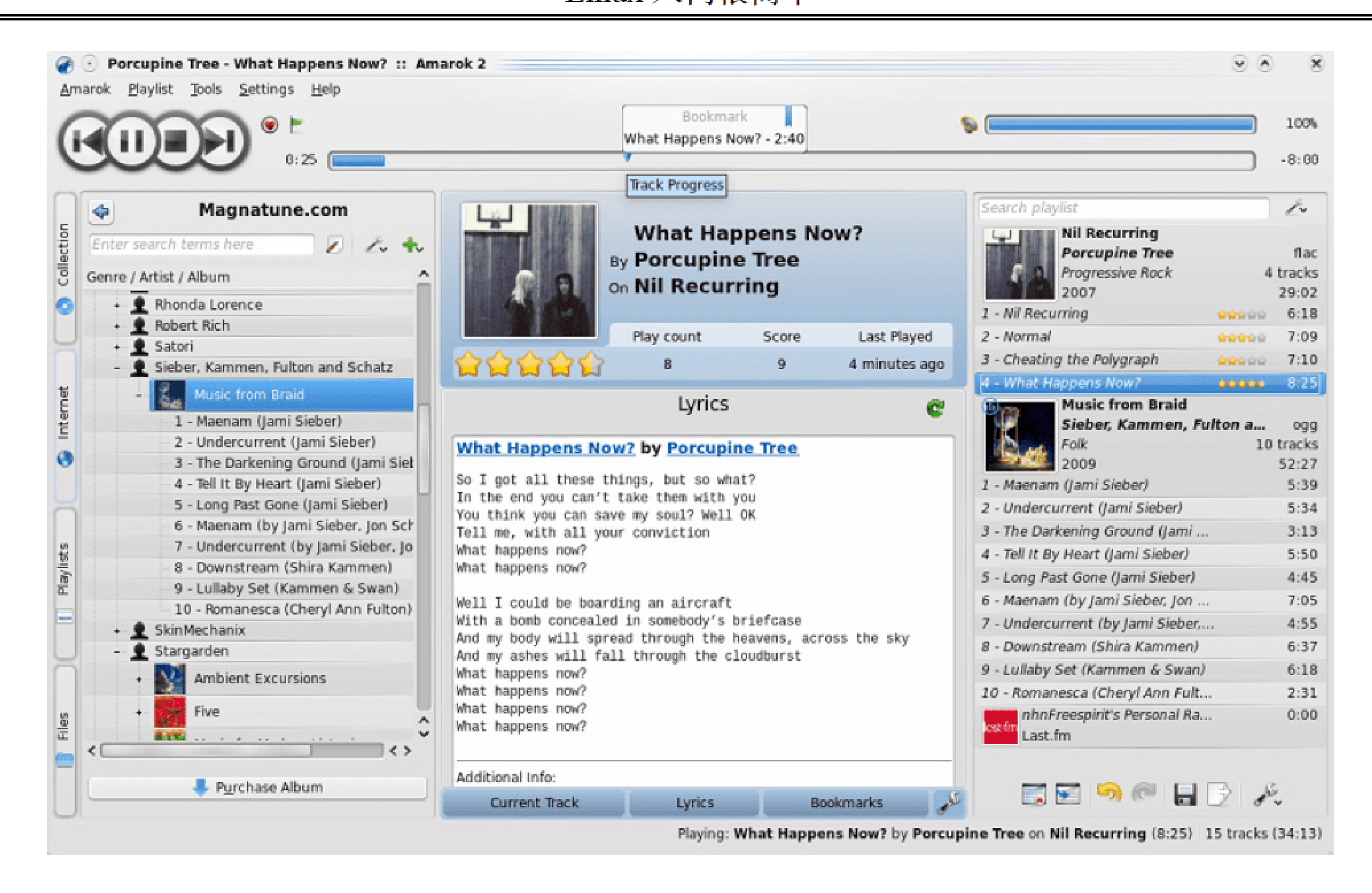


图 4.38 Amarok 界面

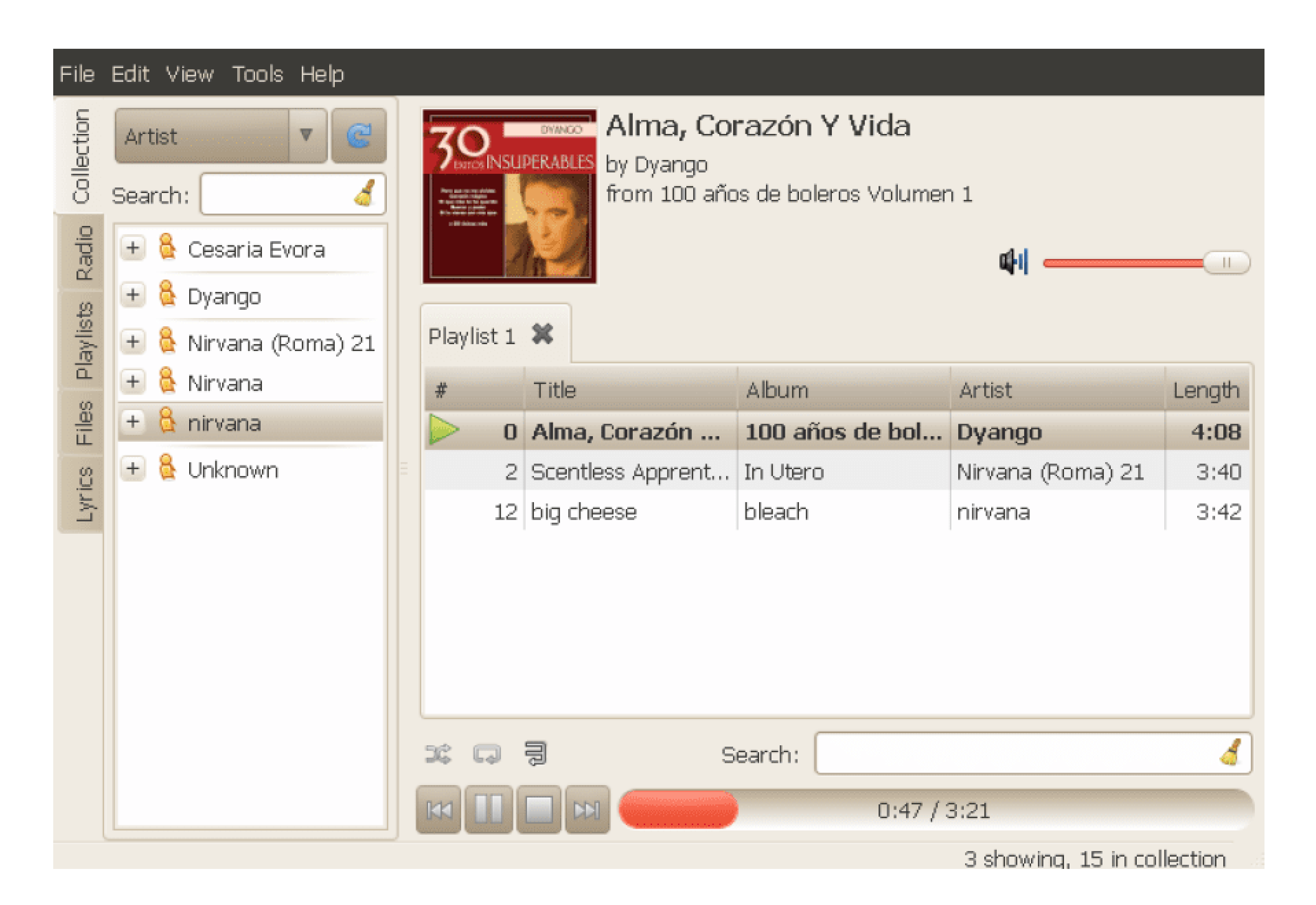


图 4.39 Exaile 界面

【梦幻的 SongBird】

说到梦，图 4.40 中的这位 SongBird 似乎是播放器里面最梦幻得了。梦幻得让人都不知道是不是该把他叫做播放器。他很苹果，很 iTune，很不像一个音乐播放器。但是他却真真切切地如一只青翠的小鸟般在吟唱着清脆的音流。他还集成一个浏览器，让你伴随着袅袅之音徜徉在浩浩网际。当看到网页上有优秀的音乐时，他还可以帮你将它收入囊中，伴你左右。



图 4.40　SongBird 界面

除此之外，还有 Banshee、Juk、LMP、Minirok、Xnosie……太多太多了，一时介绍不完。不知道今天是不是受懒蜗牛同学和 MM 的气氛传染，我的程序似乎跑得有些缠绵，哎，大约该把自己重启一下了吧。

4.3.4　扩展阅读：解码器与硬解码

本节提到了视频文件有编码，播放器要用解码器。这个视频文件都是有一定的编码方式的。比如大家都听说过 MPEG 吧，就是 Moving Picture Experts Group，动态图像专家组，听这名字本来是用来指代一小撮明白真相的群众的，不过后来这一小撮群众发布的标准被广泛使用，于是 MPEG 就成了指代这一小撮群众定义出的那一大撮标准的名词了。

【编码，从 MPEG-1 到 MPEG-4】

MPEG-1 是小撮群众在 1992 年定义出的一个标准，是一种视频和音频的编码方式。大家记得以前的 VCD 不？VCD 光盘上的视频和音频用的就是 MPEG-1 这种编码标准。而 MPEG-1 标准中关于音频的部分——MPEG-1 Layer3，更是成为互联网上及大家口袋里最常见的音频标准——MP3。

后来，1994 年，这一小撮明白真相的群众又发布了 MPEG-2 标准。MPEG-2 向下兼容 MPEG-1，并增加对隔行扫描的支持，被应用于有线电视，还有 DVD 的音频视频编码。

再后来，这一小撮群众又开发了 MPEG-3，注意 MPEG-3 跟我们的 MP3 没有任何关系，而且，MPEG3 最终没有得到很好的应用，因为当时人们发现 MPEG-2 足够了，MPEG-3 并没有提供足够好的改进。

而 1998 发布的 MPEG-4 就不一样了，它可以让视频文件的体积更小，压缩率更高，因此得到了广泛的使用。现在市场上卖的 MP4 播放器，就是用来播放 MPEG-4 压缩的视频文件的设备。所以，MP4 跟 MPEG-4 有关，而 MP3 跟 MPEG-3 无关。

【软件解码】

说了这么多，回过头来说说解码。

视频文件都进行了一定的编码，比如 MPEG-2，或者 MPEG-4。就是说这个视频文件里面的东西都是一大堆乱七八糟的数字，要想看这个视频文件，就得解码，也就是根据这一大堆数字算出应该显示的一帧一帧的图像，并且把这些图像连续播放起来，从而还原成视频。那么这个解码的过程就要靠 Mplayer 老先生了。

老先生有很多的解码器，也就是有很多的说明手册，上面写了每种编码格式的文件应该怎么计算，怎么解码。那么以前没有硬件解码的时候，Mplayer 老先生是怎么做的呢？首先，拿到一个视频文件，然后看看是什么编码的，对着自己的手册，开始解码。解码的过程就是计算的过程，计算需要用什么？那位同学回答了，得用 CPU 啊。于是 Mplayer 一手拿着手册，一手拎着数据找到我，请求使用 CPU（我是操作系统嘛，软件要用 CPU 得跟我申请）。我说，好的，你就排在 GIMP 的后面，等他用完了你用。过一会儿 GIMP 用完了 CPU，Mplayer 过去开始拿 CPU 按照手册上写的算法算他那堆数据。最后算出来，得到了几张图片，就转身把图片给图形部门，让他们去显示。然后再从那个视频文件里拿一些数据，再来排队等着用 CPU。

由于视频文件的计算量都很大，尤其是高清视频尤其大，因此为了保证蜗牛看的电影不变成带旁白的幻灯片，我就要尽可能地让 Mplayer 多用 CPU，来保证他能顺利地解码。于是，每次 Mplayer 一播高清视频，CPU 就总被他占着，搞得别的程序都抱怨。

【硬件解码】

如果他终于学会硬解码了，情况就好多了。当然，光他学会硬解码也不行，关键是显卡也得支持，而且驱动还得装好才行，不过这些咱现在不讨论，先说 Mplayer。

会了硬解码之后怎么样呢？在播放视频的时候一手拿着手册，一手拎着数据找到我，跟我说要用用显卡。可不是 CPU 了啊，改用显卡了。于是我就很乐意地让他去用了，反正别人也用不着，让他自个玩去吧。于是他就去用显卡算去了。用显卡算和用 CPU 算不一样，CPU 虽然强大，虽然啥都能算，但是要自己手动算。就是说自己要知道算法（对于 Mplayer 来说，算法都在解码器上写着呢），比如要算出一帧的视频来，要先用第 1 个数加上第 2 个数，再用结果乘以第 3 个数之类的。这里加啊，乘啊，都是用 CPU 算的，但是中间的过程是要软件（也就是 Mplayer）自己控制的。可是用显卡解码就不一样了，人家那东西是专门解视频的啊，所以你只要把数据放在里面，直接就能给你算出一帧帧的画面来。全自动啊！于是 Mplayer 不但不用跟别的软件抢 CPU 了，而且解码的速度还快了不少。

提示： 显卡上的 GPU 在计算并行计算方面比 CPU 更加强大，因此硬解码可以获得更快的速度。

4.4 我的生活色彩

今天一起床就接到了一个任务，听起来还挺轻松，一般胡同里大妈大婶的，经常做这项工作，并且乐此不疲，那就是——串门。不过我去串门可不是聊天去的，我是去做搬运工，要把一些图片复制到我们 Ubuntu 系统的硬盘里。估计懒蜗牛同学是想处理照片了，这

回，图像处理部的那几个软件有事干了。

4.4.1　从复制照片开始

能够有机会去隔壁 Windows 7 那屋里，看看他和他的同志们，我还是很高兴的。

【从邻居家复制照片】

推门进入 Windows 7 的房间，里面所有的软件都在睡觉。我看了一下，这个房间是 NTFS 格式的。想想也是，Windows 7 只能装在 NTFS 格式的硬盘上。要说以前，我们 Linux 是不太能读懂 NTFS 格式的磁盘的，毕竟是微软私有的格式，我的前辈们基本上只能勉强自 NTFS 的磁盘上读取东西，往里写是不行的。不过自从 Canonical 学校为我们增加了一本 ntfs-3g 教材以后，读写 NTFS 就都不在话下了。

提示：目前 Linux 虽然能够读写 NTFS 分区，但如果 NTFS 分区中设置了加密或压缩属性，则可能会发生异常。

不过虽然能够读懂，但是我自己是不会用这个文件系统的，我会用很多其他的文件格式，比如 ext2、ext3、xfs、jfs、reiserfs、ufs、zfs 等，各有优势，我现在的屋里使用的是非常强大的 xfs 格式，至于怎么强大，以后慢慢细聊，现在我要干活了。

来到 Windows 7 的屋里后才发现，满地都是碎片啊。别误会，这碎片并不是打架摔的，而是磁盘碎片。是这样，Windows 7 使用的文件系统比较容易产生磁盘碎片，隔三差五地就需要整理，可能是我们的蜗牛同志比较懒吧（要不怎么叫懒蜗牛呢），所以这屋里有很多碎片没有整理。不管这些了，赶紧干活，搬东西。按照懒蜗牛的指示，我找到了那些照片文件的目录，里面还有很多的子目录，叫什么 bizhi、美女、风景等，还有个目录，叫做“.隐藏”，这里面是啥呢？打开一看，还有一个目录，叫做 XX 门……

提示：Linux 读取 NTFS 分区时的速度可能比 Windows 慢一些。

【从移动存储设备复制照片】

总算把所有照片搬回来了，还没喘口气呢，我们屋里那扇 USB 门上的灯又亮了。

这扇 USB 门是我们与外界交流的一个接口，每次门上的红灯亮起，就说明有东西接到 USB 上了，我就得去打开门看看。有时候门外是一个小集装箱似的屋子，很小，一般只有几百个 MB 到几个 GB 大小，里面也像我的屋子里一样放着一些文件。这个时候一般用户就是要让我搬东西，不是把小屋子里的往大屋里搬，就是从大屋往小屋挪。有时候一开门，外面不是一个屋子，而是一台设备，比如是一个鼠标啊，或者是个摄像头什么的，那就是让我去操作这些设备了。不过总的来说，还是往 USB 门外接小集装箱屋子的情况多，这种小集装箱式的屋子叫做 U 盘，也有更大一点的，叫做移动硬盘。每当 U 盘接进来的时候，我就把它当作我屋子的一部分来用。

提示：插入 USB 设备后可以通过 lsusb 命令查看是否正确识别到该设备。

【挂载的概念】

这回懒蜗牛同学接到 USB 上的又是一个集装箱式的空间，还比较大，4 GB。我仔细看了看，结果没有发现这个设备的名字，于是我就直接给他起个名字，并在门上挂上了牌

子：/media/4.1G。

就像我屋里的/home 分区一样，我屋里的所有分区，都必须有个牌子。这个挂牌子的过程，用我们的专业话说，叫做挂载，大家装系统的时候应该都听说过。挂载，就是挂牌，就是在某一间屋子的门口挂个牌子，起个名字，干活的时候就好说话了。直接说屋子的名字就可以了。

名字，或者说牌子，就是个标志，是可以随便换的。比如有个分区被挂载到了/home/目录，也就是说，有个屋子 A，被挂上了/home/的牌子。那么用户说要看看/home/下都有什么，我就会把那个屋子 A 里面的东西列出一个单子来给用户看。回头可能又把别的分区挂载到/home/下了，很简单，就是把/home/那个牌子从 A 房间门口摘下来挂在 B 房间门口而已。用户再说要看看/home/下都有什么，我就该把屋子 B 里面的东西列表来给用户看了。没啥经验的用户可能会大跌眼镜：哇塞，怎么我的/home/下原来那些东西都没了啊？都哪去了啊？殊不知，其实原来那些东西还在 A 屋子里好好地放着，只是现在 B 房间改叫/home/了而已。

提示：移动存储设备插入系统后，系统会自动挂载。如果挂载失败可以手动运行命令挂载：

```
sudo mount /dev/<设备名> /media/<挂载点>
```

其中，设备名即为 sdb1 之类的设备文件，可以通过 fdisk –l 查看。

4.4.2 管理照片的 F-spot

扯远了，不好意思。回来说这回接上的这个 4 GB 的屋子。接上之后，还没等用户发话，我先去里面查看了一下，一看全都是些 JPG 格式的文件，我是看不懂这些文件的，但是我知道有人能懂，于是赶快通过图形界面那哥儿几个报告用户：您插进来的这个里面貌似全是照片，是不是要我给您找来 F-Spot 处理呢？提示如图 4.41 所示。



图 4.41 插入存有照片的存储设备时的提示

F-Spot 是我们这里的一个管理照片的程序，他可以帮用户把移动设备（就是那种集装箱似的小空间）里的照片导到硬盘里，并且按时间分门别类地管理好。用户想要去年 3 月的照片，他马上能够给找到；要前年 5.1 假期的，没问题；要看宋朝的……给把铲子自己挖去吧。

除此之外，他还能够汇报照片的信息，比如用什么相机照的，照的时候用的光圈，快门，ISO 都是多少等信息。这例不是 F-Spot 厉害，能够光看照片就知道是用什么相机照的，而是因为相机在照这张照片的时候就把各种参数写进照片文件里了。

🔔提示：数码相机拍摄的照片包含 EXIF 信息，记录了拍摄时的情况。除 F-Spot 这样专门管理照片的软件外，在 Gnome 中直接查看照片文件的属性，选择图像标签就可以看到。如图 4.42 所示。



图 4.42　查看照片 EXIF

懒蜗牛似乎对 F-Spot 的能力还不大熟悉，于是没有让我去叫醒 F-Spot 来导入照片，而是自己手动把 U 盘里面的照片复制到了那个放照片的目录。复制完之后，懒蜗牛看了看图片目录的文件夹：bizhi、美女、风景，还有从 U 盘复制过来的“静物”，嗯……都齐了。接着，就下达了卸载 U 盘的命令。

🔔提示：卸载 U 盘可右击桌面上 U 盘的图标，选择“卸载“或“安全移除驱动器”。“卸载”只卸载 U 盘上的文件系统，并不断电，处理方式类似 Windows 7。“安全移除驱动器”包括卸载和断电两个动作，处理方式类似 Windows XP。

有人可能会问懒蜗牛同学怎么没看见那个叫“.隐藏”的目录，以及里面的 XX 门照片？这一点要解释一下了，在 Windows 7 那里，文件和文件夹的属性中有一项是隐藏，隐藏的文件一般是看不到的。而在我们 Linux 这里，文件是没有隐藏这个属性的，但是也有办法让一个文件或文件夹在默认情况下不显示出来，那就是给这个文件起一个以“.”开头的名字，那样，无论自终端里执行 ls 命令，还是用“文件浏览器”来查看，都看不到这个文件。不知道懒蜗牛是有意的还是无心的，反正那个目录的名字既然叫做“.隐藏”，他自然就看不到了。

🔔提示：要想看到也容易，在终端里用 ls -a，在图形界面的文件浏览器里就按 Ctrl+h。

照片复制进来后，懒蜗牛让狐狸妹妹去找找好用的照片管理软件，终于从狗狗那个无

敌的搜索引擎里知道了，原来我们 Ubuntu 本来就带有 F-Spot 来管理照片。绕了一圈终于绕回来了，懒蜗牛同学下令：运行 F-Spot。哎，你说你刚才复制照片的时候就用他多好。

F-Spot 被我叫醒之后，赶快进入工作状态，由于是第一次运行，他上来就报告：现在还没有导入任何照片，请选择要导入的照片，如图 4.43 所示。

这话说得在理啊，他就是个管理照片的，你得赶紧跟他说让他管理哪些照片啊，要不然启动了他也没事干。可是懒蜗牛同学似乎对这个“导入”有些迷茫，心说，我把照片都放在“图片”那个目录下了啊，你不会自己找找么？得，你让我导入，我就导入把，于是他在 F-Spot 给出的导入窗口中选择了“选择文件夹”，同时他还看到了“（未检测到相机）”这个不可选的项，看来如果有相机接进来，是可以直接从相机中导入照片的。懒蜗牛选择了图片目录下的“bizhi”、“美女”、“风景”等几个文件夹来导入，F-Spot 立刻工作起来，不一会儿，懒蜗牛就在 F-Spot 的主界面上看到了自己导入的那些图片，并且已经按照时间顺序整理好了。选中一张图片，还能在左侧显示详细信息，如图 4.44 所示。




图 4.43　F-Spot 导入图片界面

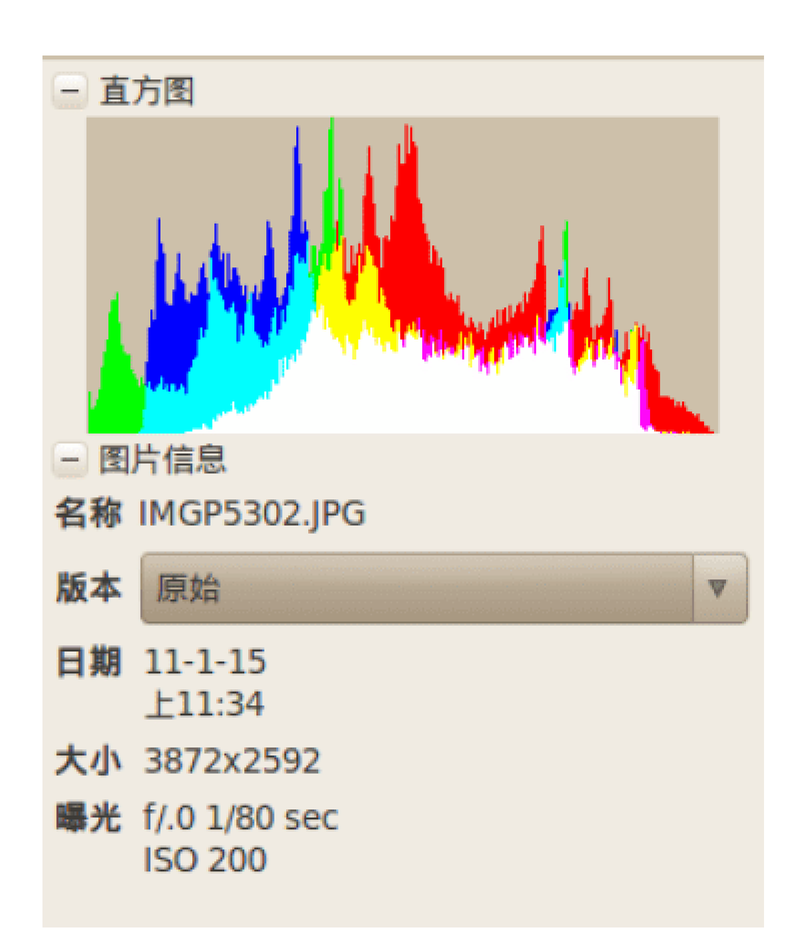


图 4.44　F-Sopt 中显示图片信息

研究了一会儿，懒蜗牛觉得这个软件还算比较好用。关了之后，打开了图片文件夹，一打就开愣了——怎么在图片文件夹里面多了个“照片”文件夹呢？赶紧进入照片文件夹看看，发现里面以拍摄时间创建了很多级的目录，最终的目录里就是原来那些 bizhi、美女文件夹里的照片。原来，F-Spot 的正常使用流程是从相机或者存储设备中“导入”照片，这个导入的过程其实说白了就是把照片从这些设备中复制到本地硬盘上，F-Spot 会在图片目录下建立照片目录，所有导入的照片就都放在这底下，可是蜗牛不知道啊，结果就把本地磁盘上的图片导入了一遍，于是每个图片就都有了两份，图片目录下一份，照片目录下一份。这一下让懒蜗牛觉得很不爽：这是什么破软件啊，原来是这么个导入法啊，太弱智啦！

4.4.3　系出名门的 Picasa

暴走一阵之后，懒蜗牛重新坐下来叫过狐狸妹妹，让她再找找管理照片的软件。其实我们 Linux 下用来管理照片的软件确实少一些，但是不乏精品。首先，F-spot 能力就不错，

只是懒蜗牛不大习惯他的方式，呵呵。另外还有 digikam，是 K 派那边强大的照片管理软件，比 F-Spot 还要强大，不过这话可不能当着 F-Spot 说，要不又吵起来了。

【身份神秘的大师】

懒蜗牛找了一会儿，终于做出了自己的选择，叫来超级牛力，安装软件。只听得超级牛力高喊着：“本 APT 有超级牛力……”就跑出去了。我问 ibus：“刚才用户给超级牛力输入了什么？”

“报告头儿，是 Picasa。”

“星爷，查查这啥意思。”星爷就是星际译王，一个字典。

“这个嘛……英法美德俄日意奥的语系里都没这个词。不过有一个长得比较像的。”

“什么？”

“Picasso，毕加索。”

“狐狸，去网上查查这是个什么软件。”

狐狸妹妹很快回来汇报：“是个照片管理软件，还找到个截图，您看。”我看了一下，还挺漂亮，就是图 4.45 所示这个图。

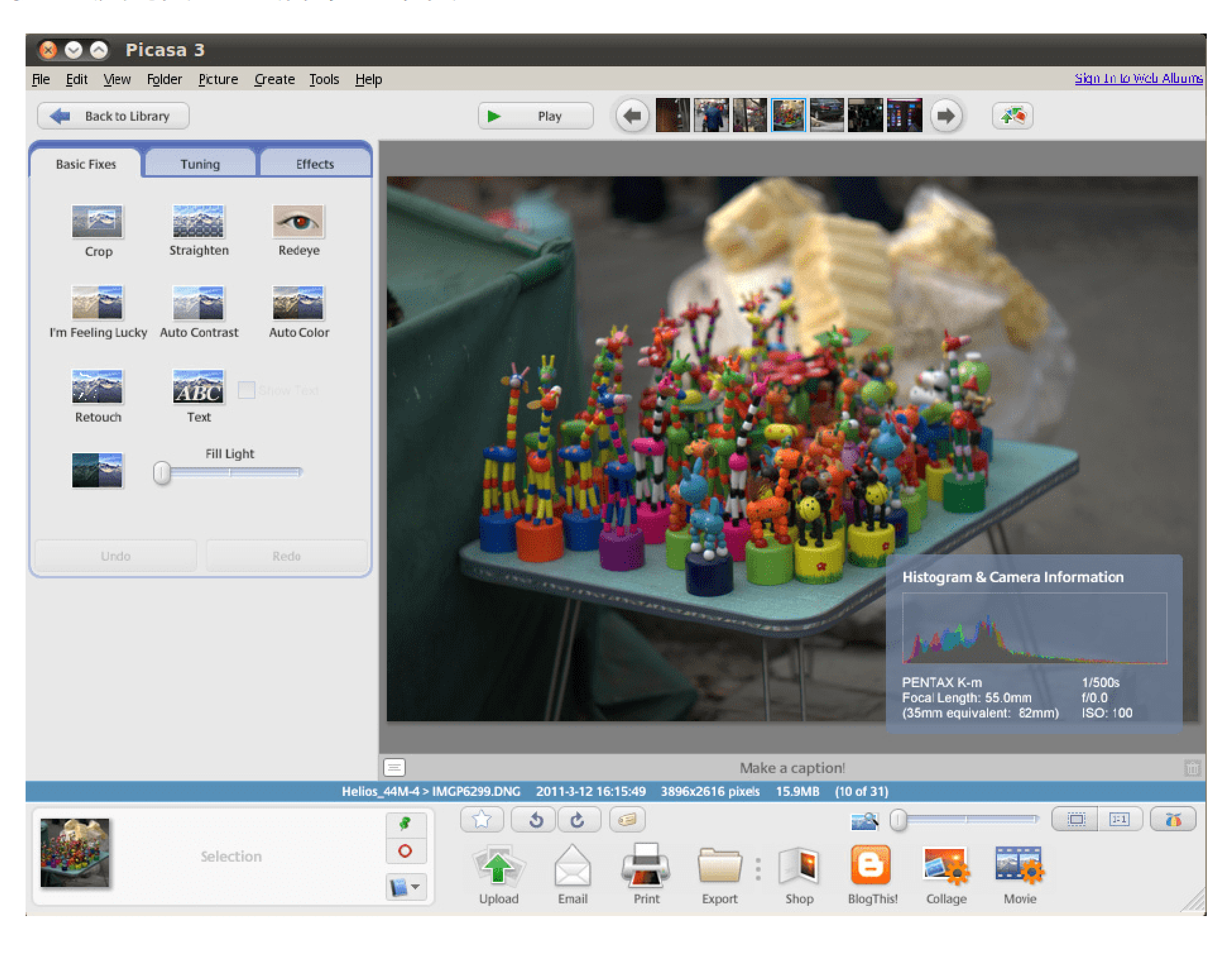


图 4.45　Picasa 运行界面

我沉思了一下说：“哦……看来懒蜗牛是要换个管理照片的软件了。”

F-Spot 不服道：“管理照片？难道我刚才的表现不好么？”

我只得双手平摊做无奈状……

数分钟后，超级牛力归来，带来了一个穿得花花绿绿，很有艺术气息的家伙。等一下，这哥们儿身上的颜色怎么好像在哪见过？我过去上看下看左看右看，怎么看都不像我们这的人。于是我叫来了 file。file 可不是一个普通的文件，而是一个程序，一个用于判断文件类型的程序。他可以判断一个文件是什么类型的文件，当然也能判断可执行的程序。

他可不是根据扩展名来判断，叫“.jpg”的就是 jpg 文件，叫“.txt”的就是文本文件，这种功能，连傻子都会。file 的功能要强大得多，他是根据文件的内容来判断的。一般一个文件都会有个文件头，来说明这个文件的类型。比如 JPEG 类型的图片文件，他的文件开头的两个字节肯定是 FFD8（16 进制），而 GIF 文件的文件头就是 4749463839，也就是 GIF89 几个字的 ASCII 码。二进制程序也有类似的特征码。于是，我让 file 赶快去看看这个“毕加索”（以下“毕加索”或“毕大师”特指 Picasa）到底是个什么程序。file 把毕加索上上下下地检查了一遍，得出结论——这是个 Windows 的 EXE 格式的程序。

提示：file 命令使用方法，file /<路径>/<文件名>。

【带着翻译来打工】

“什么？Windows 的程序？！超级牛力啊，你别是走错了吧，怎么把 Windows 的程序领来了？”超级牛力不急不慌地摇摇头：“本 APT 有超级牛力，怎么会搞错呢，这个就是从源里找来的软件包。不过别急，本 APT 有超级牛力，这软件包可不是光毕加索一个，后面还有一个呢。”我这时才注意到，老毕后面还站着一个家伙，这人又是谁呢？没事，超级牛力那里肯定有这个软件的资料，让他查查吧。还没等我让他查呢，他已经向大家解释上了：“毕加索先生是 Windows 界成名的图片管理大师，他所在的公司，也就是狗狗哥那公司，他们公司为了惠及 Linux 世界的人们，也为了偷懒，给毕大师配上一个翻译就直接推向了 Linux 界。”哦，原来是这样，我说怎么看着这身花里胡哨的颜色眼熟呢，因为他跟那个 Chrome 的颜色差不多，都是狗狗公司的嘛。那么后面那个既然是专门负责给毕大师当翻译的，我们就叫他毕翻译吧。

毕大师和毕翻译安顿好之后，懒蜗牛没有立刻把他们叫起来干活，他似乎有些什么别的事情，装完了 Picasa 就离开了。但是毕大师及其翻译并没有消停，俩人先后被 F-Spot 拽进内存，一场争斗又要开始了。

F-Spot 是一直跟着我的软件，我对他的脾气很了解，有点太直，他这种元老级的软件被用户换下来自然不爽。一直以来都是他负责管理照片的，论能力他自己觉得不输给任何人。现在蜗牛刚用了他一次就找来这么个功能差不多的家伙来代替他，这不是明摆着砸他 F-Spot 的牌子么。要是以后让这个 Windows 的程序代替了，我们 Linux 程序的脸面还往哪搁？于是 F-Spot 决定，为了荣誉，向毕加索挑战！

只见 F-Spot 跑到被拉进工作间的毕大师面前说：“大师既然出身名门，本领自该不弱。在下不才，愿在大师面前领教几招粗浅功夫，不知道大师可肯赐教否？”只见毕大师的表情如平静的湖水般，并没有因 F-Spot 的挑战而激起一丝波澜，只是面容祥和地扭过头对翻译说：“What did he say?”呃……

经过毕翻译的解释，毕大师明白了 F-Spot 的意思，微微一笑，做了个请的手势。我一看，得，又斗上了。

【决战毕加索】

要比就从起床开始比！F-Spot 和毕加索及毕翻译重新回到硬盘睡觉，然后我去叫来了 time 同志。time 是一个用于计时的命令，咱以后再说，先看比赛。随着我的一声号令，time 开始计时。F-Spot 蹦起来后牙也不刷，脸也不洗（废话，一个软件，有牙么？），迅速地从硬盘飞奔进内存。再看那边，毕翻译先迅速跑进了内存，然后再扭头去叫醒毕大师——因为毕大师听不懂我们的话，所以无论我们怎么喊都是叫不醒他的，只能先叫醒翻译，再

由翻译去叫醒他。这样一来，时间自然慢了不少，对于起床速度，F-Spot 完美胜出。

提示：time 命令是计算程序运行时间的命令，其实是无法单独计算从硬盘读取到内存这段时间的。

双方起床已毕，相向而立，只见 F-spot 掏出两张一模一样的照片，照片上是一个人像，似乎是晚上照的，眼睛如含着血泪般发出令人不寒而栗的红色。只见 F-Spot 把一张照片扔给毕大师，另一张贴在自己这边，双掌运足力气，瞄准照片中人的双眼大喊一声：嗨！立时，照片上人的红眼不见，翻了白眼。另一边的毕大师微微一笑，拿起自己这边这张，单掌向前一推，一股掌风直逼那人双眼，只见掌风过后，那人双眼渐渐恢复成正常颜色。

F-Spot 不等毕大师打完那掌，又拿起照片推拳运功，只见那本是夜里的照片亮如白昼。毕大师也不示弱，将照片抛向空中，双手一抖，一道劲风吹过，再看落下来的照片时，也已经比原来明亮不少。

F-Spot 又对照片连续发力，打出 3 招，依次改变了照片的对比度、色调和饱和度。毕大师口念咒语："That's easy……"只出一招，双手间出一道白气，就把照片的亮度、对比度、色调、饱和度，都改到合适的状态。毕翻译在旁边解释道："这招乃是毕大师的独门秘诀，叫做'手气不错'！"毕大师微微点头，一扬手，只见那张修改好的照片激射而出，直接从网口飞了出去，发布到了 PicasaWeb 网站上。屋内众人顿时为 F-Spot 捏一把汗，大师叫 Picasa，那么这 PicasaWeb 网站，明显是人家地盘啊，F-Spot 能搞定么？哪知道 F-Spot 不慌不忙，也把照片往网口一扔，把照片同时发布到了 Flickr、PicasaWeb 等多个网上相片储存空间里，真是棋逢对手啊。

正闹腾着呢，懒蜗牛同学已经回来了，听声音后面似乎还有一个人，她一说话我就听出来了，就是上次那个 MM。

4.4.4　Gnome 之眼

懒蜗牛同学似乎是和 MM 欣赏照片来的，只见他打开了 Picasa，毕大师和毕翻译跟 F-Spot 斗完了还没来得及回硬盘呢，正好就直接投入工作了。蜗牛还夸 Picasa 启动很快，恨得 F-Spot 牙根痒痒——他启动能不快么，压根就在内存里呢。毕大师第一次启动，要扫描用户的整个目录，懒蜗牛也没在意就直接允许了，然后毕大师就开始一个目录一个目录地搜索，先是"bizhi"、然后是"美女"、之后的"风景"、还有"XX 门"——等等！当着 MM 面看这个不大好吧？

【隐藏暴露的危机】

当初把这个目录复制过来的时候还以为是懒蜗牛同学为了隐藏，故意起名以"."开头呢，合着他自己也没注意到这个目录啊。也是，复制过来之后就隐藏了，他当然看不见。F-Spot 是需要手动导入照片的，导入哪个目录才能看到哪些图片，不导入的看不到。毕大师可不一样，他可是搜索你目录下所有的图片，不管什么隐藏不隐藏。话说回来了，毕大师本来就是 Windows 的软件，他们可不知道"."开头的软件是隐藏的啊。结果毕大师很实在地把那个".隐藏"目录下的 XX 门目录的图片也都给扫描了。

毕大师一边扫描，一边在屏幕的右下部分显示出当前扫描到的图片的缩略图。虽然每

张图片都是一闪而过，懒蜗牛似乎还是看到了，赶快把毕大师最小化了。可是毕大师似乎并没有理解用户的意图，主窗口虽然最小化了，但是依然在屏幕的右下方显示着正在扫描的图片。不过所幸毕大师扫描得很快，一转眼的功夫已经开始扫描下一个目录了，懒蜗牛同学的心才算放下去一点，这时候听那 MM 说：照片呢？快让我看看吧。

懒蜗牛似乎迟疑了一会儿，到底是该打开毕大师来浏览图片还是去那个目录下一张一张手动打开看呢？经过了复杂的思想斗争后，终于决定——把毕大师关了！然后打开文件浏览器，找到存放照片的那个文件夹打开来看，安全第一！

来到存放着照片的文件夹，只听 MM 惊呼："咦？照片都是这样显示的啊，真先进。"一看就是只用 Windows XP，连 Windows 7 都没用过的家伙。蜗牛得意地点点头："那是那是。"心里可一直打鼓："这么手动一张一张看，会不会被鄙视连个 ACDSee 那样的软件都没有呢？"但反正也没有退路了，他也就只好双击了一张照片。按照默认的设置，我去叫醒了 Eye of Gnome，中文叫做 Gnome 之眼。

【临危受命的 G 大眼】

这家伙是一个用来浏览图片的软件，功能上相当于 Windows 7 那里自带的图片查看器。Gnome 之眼（咱就简称他 G 大眼吧）支持基本上所有常见的图片格式，可以对图片进行缩放和旋转。懒蜗牛看到了 G 大眼的界面，如图 4.46 所示，虽然简洁，倒也像那么回事，这才放心了。他一边和 MM 评论着照片，一边按向右的按键查看下一张。这个目录下的照片是从 U 盘复制过来的那堆照片之中的一部分，似乎是懒蜗牛和 MM 出去玩的时候照的，俩人一边看，一边评论。看到一张照片有些问题，里面的人都红眼了，这是晚上照相的时候经常遇到的。懒蜗牛顺手单击 G 大眼界面右上角的"编辑图像"按钮，G 大眼就叫来 F-Spot 来对图像进行编辑。



图 4.46　Gnome 之眼界面

刚才 F-apot 跟毕大师比武的时候咱也看见了，修正个红眼对 F-Spot 来说不在话下。这一下 F-Spot 可算是有了一种看见伯乐的感觉了。G 大眼和他都是 Gnome 自带的标准软件，是 G 帮的招牌，俩人配合起来天衣无缝。F-Spot 在左侧列出了可以对这张照片进行的处理，如图 4.47 所示。蜗牛选择了"降低红眼"，很方便地就修复了过来。



图 4.47　调用 F-Spot 处理照片

继续往下看，有的照片是相机竖着照的，需要旋转一下。这事就不必惊动 F-Spot 了，G 大眼自己就给解决了。又看了几张，懒蜗牛觉得一张一张往下翻太麻烦，想起在 Windows 7 下用 ACDSee 的时候可以按 F5 键自动播放，于是抱着碰碰运气的心态，也在 G 大眼的界面上按了一下 F5 键，果然 G 大眼就进入了全屏播放的状态。这种状态下，手动翻页也可以，如果不翻页，到一定时间会自动显示下一张。

【危机过去】

说说笑笑中，时间很快就过去了。现在时间是晚上 19 点 00 分 00 秒，距离晚饭还有 1412 秒。今天懒蜗牛同学与 MM 进行了亲切的会谈。会谈中，宾主双方就哪些照片该删，哪些照片照虚了等原则性问题，达成了高度一致的共识。MM 高度赞赏了懒蜗牛同学操作电脑的能力。懒蜗牛也表示将继续坚持两个系统一起抓，两手都要硬的原则，并对 MM 一贯相信世界上只有一个系统，微软是计算机不可分割的一部分的想法表示万分哀悼。会谈中，懒蜗牛同学还指出，Ubuntu 是 Linux 中的一员，在当今世界形式下，应团结 Linux 世界中所有发行版力量，为把自由软件建设成为富强、繁荣、发展的共产主义新软件而努力奋斗。会谈结束后，懒蜗牛邀请 MM 共进晚餐，餐前还没忘把我关了。

晚上，懒蜗牛独自归来，一回家就赶紧把我叫醒，启动之后第一个命令就是去找来毕大师。我明白懒蜗牛是想看看那个 XX 门的图片到底是怎么回事，为什么之前一直没有注意到，却被毕大师搜索出来了。毕大师起床后在主界面按时间顺序列出了自己搜索到的图片，懒蜗牛很快找到了“.隐藏”目录下的“XX 门”目录，看见里面那一百多张图片，不禁血压上升。只见每张图片拍摄角度各不相同，色彩灯光各有差异，但上面都无一例外地有两个主要元素——一扇孤独的铁门，和一行“插插防盗门，插上不进人”的广告语……懒蜗牛泪奔而去。

4.4.5　免费的 PS——GIMP

自从 F-Spot 和 G 大眼的配合获得懒蜗牛的认可之后，F-Spot 的心里总算是平衡了不少，

不过平时懒蜗牛整理照片还是主要用毕大师，F-Spot 只是用来顺手处理一下照片。可是这 F-Spot 毕竟是专业管理照片的，不是专业处理照片的，就算他能通过安装插件来扩展对照片处理的效果，可毕竟能力有限。于是懒蜗牛通过明察暗访，终于知道了一个响当当的名字，一个更加强大的图片处理软件——GIMP。

GIMP 这家伙是个天才的美术家，是我们 Linux 界有名的图像处理软件，号称 Linux 的 PS，就是图 4.48 中这个分成了很多窗口的软件。他能够画出很多漂亮的图画来。当然，漂亮不漂亮是以人类的审美观点来说的，对我来说他制作出来的那些东西，不过是一个写满 0101001 的文件而已，和其他的文件没有什么区别。

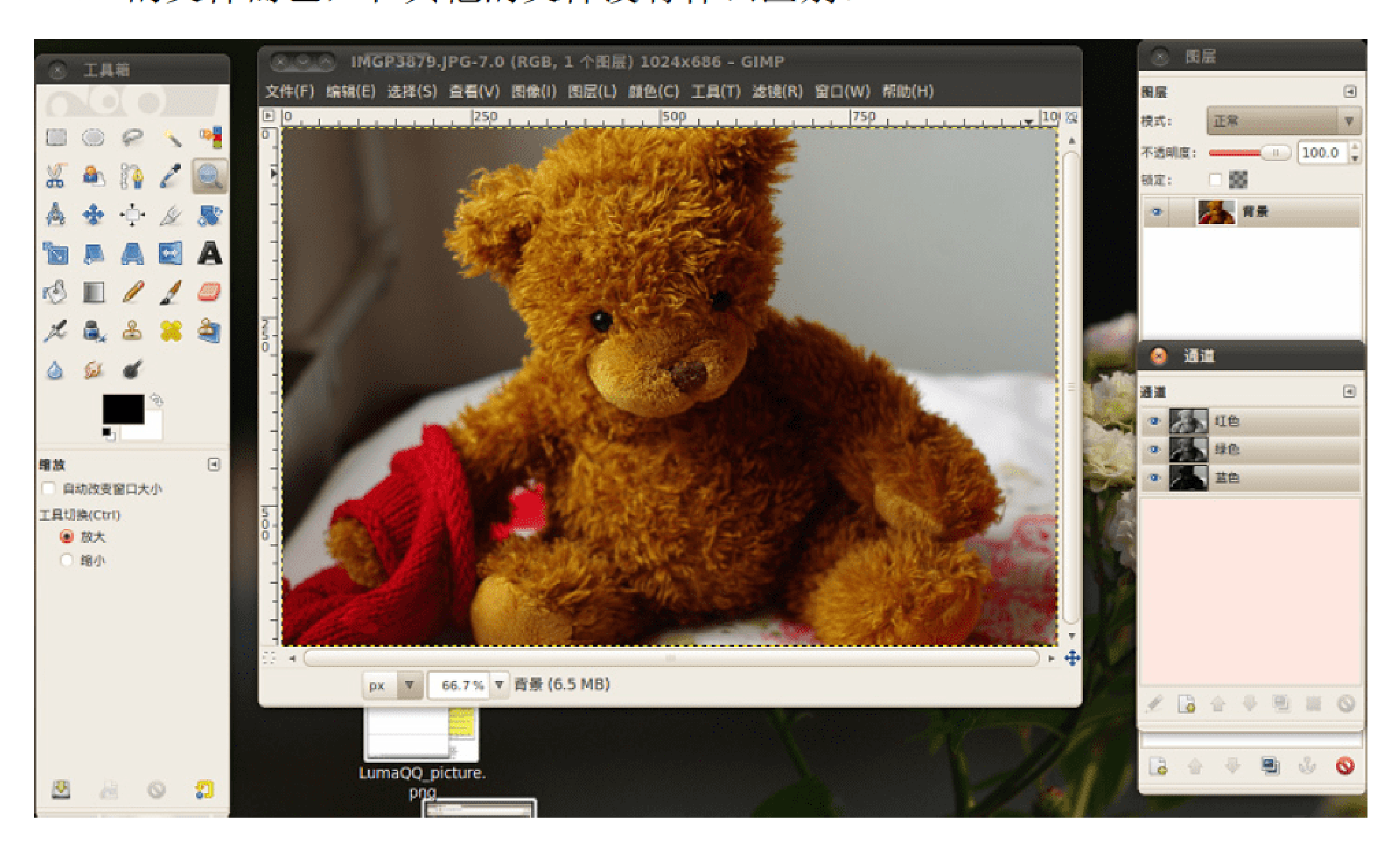


图 4.48　GIMP 运行界面

虽然他可以在一张白纸上创造出一个多彩的世界，但多数情况下他要做的工作是调整一个已经画好的图片。比如调整一下图片的亮度，色彩什么的。有时候还让 GIMP 做一些效果，比如做成油画效果的，还有做成浮雕样的，或者加个相框什么的，他都行。更高级一些的，比如在树林里画出一只华南虎啊，把图片里路过的路人甲换成小胖之类的，也都没有问题。

同时 GIMP 也像 Firefox 妹妹一样，可以安装很多的插件以实现各种不同的功能和效果。论功能，不输于 Windows XP 那里的 Photoshop（PS）。但是 GIMP 有些不大好接触，总是按自己的性子来，让用户用起来有些不顺手，不如 PS 那么易用。毕竟人家 PS 身价不菲嘛，听说一套要好几千块钱，一分钱一分货嘛。花这么多钱请回家去的软件，怎能不服服帖帖地听用户的话呢。再看看 GIMP，总是摆着一副“我就这样，爱用不用”的架势。也许就是因为他太过冷酷，原本默认安装在系统里的 GIMP 从我们这一届 Ubuntu 开始，不再默认安装了。毕竟多数用户用不到这么专业的软件。如果想用他也不难，找超级牛力装上就可以了。

懒蜗牛得知有这么个软件后，赶忙装上体验一下。他首先用 GIMP 打开了一张照片，试着调整一下亮度、对比度，拉了拉曲线，觉得不如 Photoshop 好用，论易用性自然差些，论功能，也就只相当于 30%的 Photoshop，不过也算够用了，以后就可以完全在 Ubuntu 里

处理照片了。

4.4.6　扩展阅读：磁盘碎片的产生

前面我们说到了 Windows 7 那里有磁盘碎片。什么是磁盘碎片呢？它是怎么产生的？下面我就给大家讲讲这个磁盘碎片。

【Windows 7 管理硬盘的方式】

同学们都坐好，都把手机铃声关了，小灵通调成震动，BP 机直接扔了——台都没了你还留着它干嘛。好，上课了，首先说说什么叫磁盘碎片。

磁盘，是我们程序居住的空间，我们用不同的方式对整个磁盘的空间进行管理。前面说过，包括各种方式，什么 ext4、xfs、ntfs 等。而磁盘里放的东西，就是一个一个的文件，同学们可以把磁盘想象成你家的屋子，文件就像一个个大大小小的箱子。每个箱子上面写着字，就是文件名。Windows 7 喜欢把每个箱子都紧挨着放，一个挨一个，上下左右前前后后都紧贴着。这样，看上去很规整，可以让剩余的空闲空间比较完整。有同学说，我家也这么收拾，这样很利索呀。不过，对于操作系统，这样做虽然有好处，但是也会有一些问题。

提示： Linux 对磁盘的管理机制不同，文件存放位置相对分散，因此不容易产生碎片。

【文件增大产生碎片】

比如，一开始存了一个文件，也就是搬来了一个箱子，比如叫“日记”。Windows 7 把它放在最靠墙的位置。后来又存了很多其他的文件，在“日记”文件的前前后后，左左右右，上上下下都放满了。忽然这一天，日记文件被修改了，加了点内容，就相当于往“日记”那个箱子里加了东西。可是箱子已经满了，再往里加，箱子就要增大（也就是文件大小变大，毕竟是比喻，不是真的箱子，大家不用费脑子想箱子怎么会伸缩）。可是箱子周围堆满了其他的箱子，没地方了，怎么办呢？

可以把边上的箱子挪开一点，原来的箱子就可以扩大了。可是边上的箱子要是少还好办，要是很多，还都装了铅块铸铁大理石什么的，那可就累死了。那怎么办呢？只好把新的内容放在另一个小点的箱子里，放在别处。然后还得在原来的“日记”箱子上标注上“日记（第 1 部分，第 2 部分在东墙根）”。然后在新的箱子上写“日记（第 2 部分，结束）”。日子长了第 2 个箱子也被很多箱子挤在中间后，又要编辑日记文件，这个文件又变大了，就又要如法炮制出第 3 个箱子，乃至第 4 个，第 5 个……

【碎片太多影响性能】

等到有一天，要读取这个日记文件的时候，Windows 7 就忙开了——首先，到西墙角找到日记第 1 部分，翻腾出里面的内容，然后往箱子上一看“第 2 部分见东墙根”，然后再跑到东墙根找第 2 个箱子，翻腾出里面的内容，再一看箱子“第 3 部分见大衣柜上头”，然后又搬梯子，上大衣柜一看“第 4 部分在厕所水箱后边”……等到 Windows 7 把整个日记文件读完了，也累得半死了。这种情况，就是会影响性能的磁盘碎片。好，本节课到此结束，同学们自由活动吧，那位同学，快去捡你的 BP 机去吧，说不定还能找着。

4.5 我的办公软件

今天蜗牛打开电脑，破天荒地先去叫醒了 OO 老先生，往常都是先去叫醒狐狸妹妹或者去找那个 World of Goo 玩一会儿。今天找 OO 老先生干什么呢？

4.5.1 代替 MSOffice 的 OpenOffice

OO 老先生知道是谁吧？就是之前给 SMplayer 和 Gnome Player 劝架的那位。我们喜欢叫他 OO，是因为他戴着一副很有学问的眼镜。图 4.49 所示就是他的工作照。他的全名，叫做 OpenOffice.org——相信我，这确实是个软件的名字，当然，同时还是个网站的名字。OO 老先生是我们这里的老前辈，他是我们这里的文书。用户要写点什么、做个什么报表、演示文档之类的，都找他。今天懒蜗牛同学找到他，打开了一个空白文档，写下两个大字——简历。

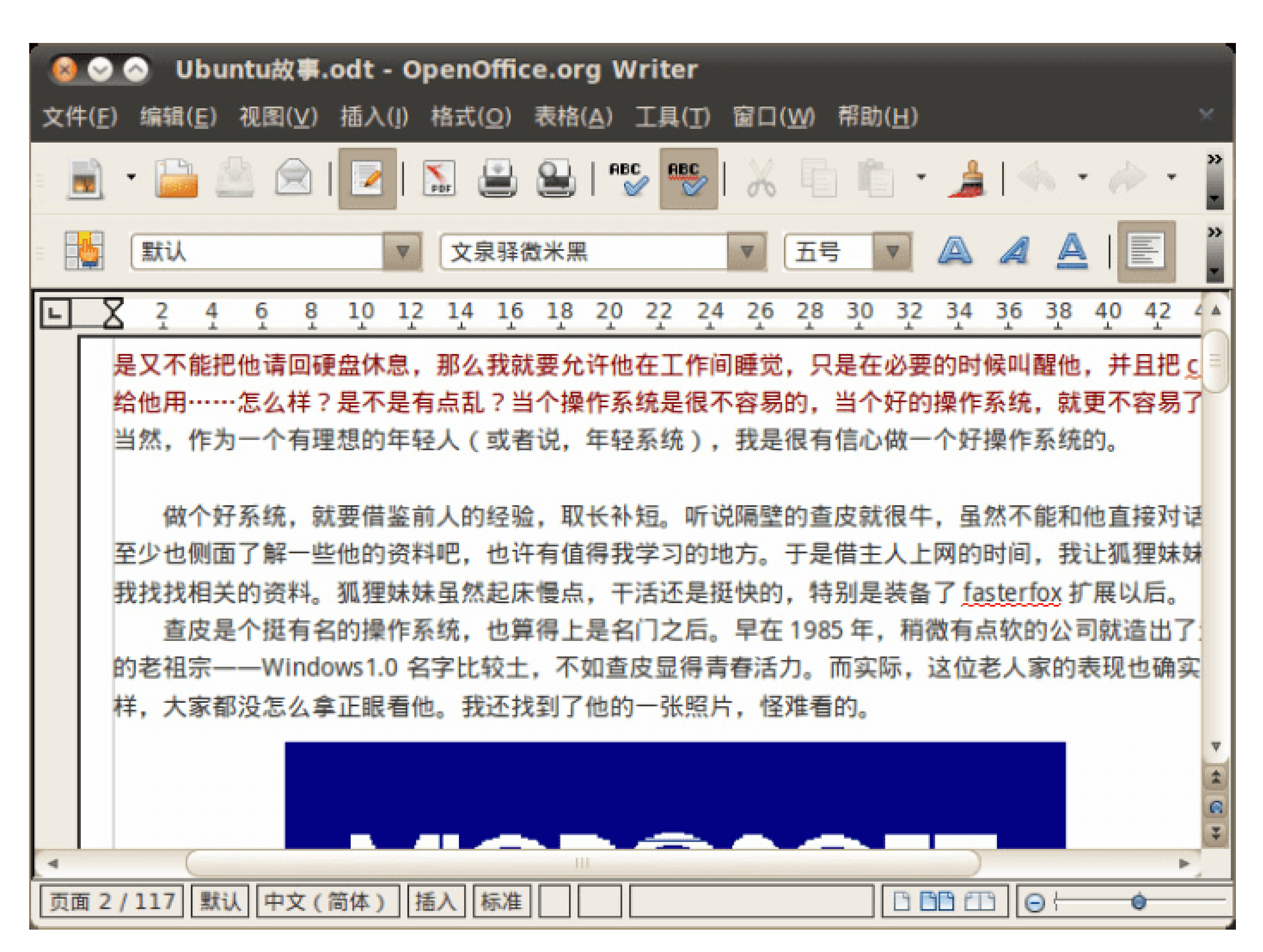


图 4.49 使用 OpenOffice 编辑文档

提示：2010 年 9 月 28 日，一些原本参与 OpenOffice.org 项目的成员，成立了一个叫做 The Document Foundation 的新团队，并创建了基于 OpenOffice.org 3.3 的分支版本——LibreOffice 3.3。

原来懒蜗牛是要找工作了啊。他们人类找工作可是个麻烦的事情，又得投简历又得面试什么的，还一面二面三面四面，五六七八面。哪像我们操作系统，看着顺眼就往硬盘上一装，简单。懒蜗牛写下了头两个字后，便呆呆地等了半天，也不知道干什么呢，等了大约 5 分钟，终于有下一个动作了——叫醒狐狸妹妹。然后他牵着狐狸就去网上找简历模板了，看来还是个新手，呵呵。要说模板，OO 老先生就有很多的模板，文字的、幻灯片的、表格的，都有。但是默认并没有安装这么多，你可以到在背后支持 OO 老先生的社区去找，

那里有很多的模板资源，地址就是这里：http://opentemplate.org/。

不过懒蜗牛同学似乎并不了解这些，他还是依靠搜索引擎才找到了一些简历模板，照着写起来。

懒蜗牛这边写着，OO 老先生在内存里一边看着一边评论："嗯……这句不合适。""这个字体不好看，怎么用这个呢？""嗯，这段不错。"没多一会儿，懒蜗牛已经凑合出了一篇中文的简历，马上还要再写一篇英文的，于是懒蜗牛又把另外一位有学问的叫醒了，他叫做星际译王，之前也出现过，我们都叫他星爷（可不是周星驰啊，以下"星爷"特指星际译王）。

这一下 OO 老先生自然还要继续工作，星爷也被调动起来——查词。那是一个词一查啊，星爷十分怀疑蜗牛的英语四级是怎么过的，心说你给我装这么多本字典自己怎么不先好好看看。可是抱怨也没用，软件的天职就是运行嘛。连查带编，总算是把英文的简历也凑出来了，尽管英国人不一定看得明白，但是有一个总比没有好。写完了之后懒蜗牛赶紧让 OO 老先生把简历存起来，老先生其实在懒蜗牛写的过程中就不断地在往硬盘存了，防止忽然掉电嘛，这都是跟 Windows 7 里面的那个 Office 学的。现在懒蜗牛要存，OO 老先生问明白了存的地址和文件名，就把文件存好了，然而存好了之后他微笑着，自言自语地说：这个早晚还得重写……

4.5.2　翻译软件星际译王

既然说到星爷，咱就来介绍介绍，就是图 4.50 所示的这位。

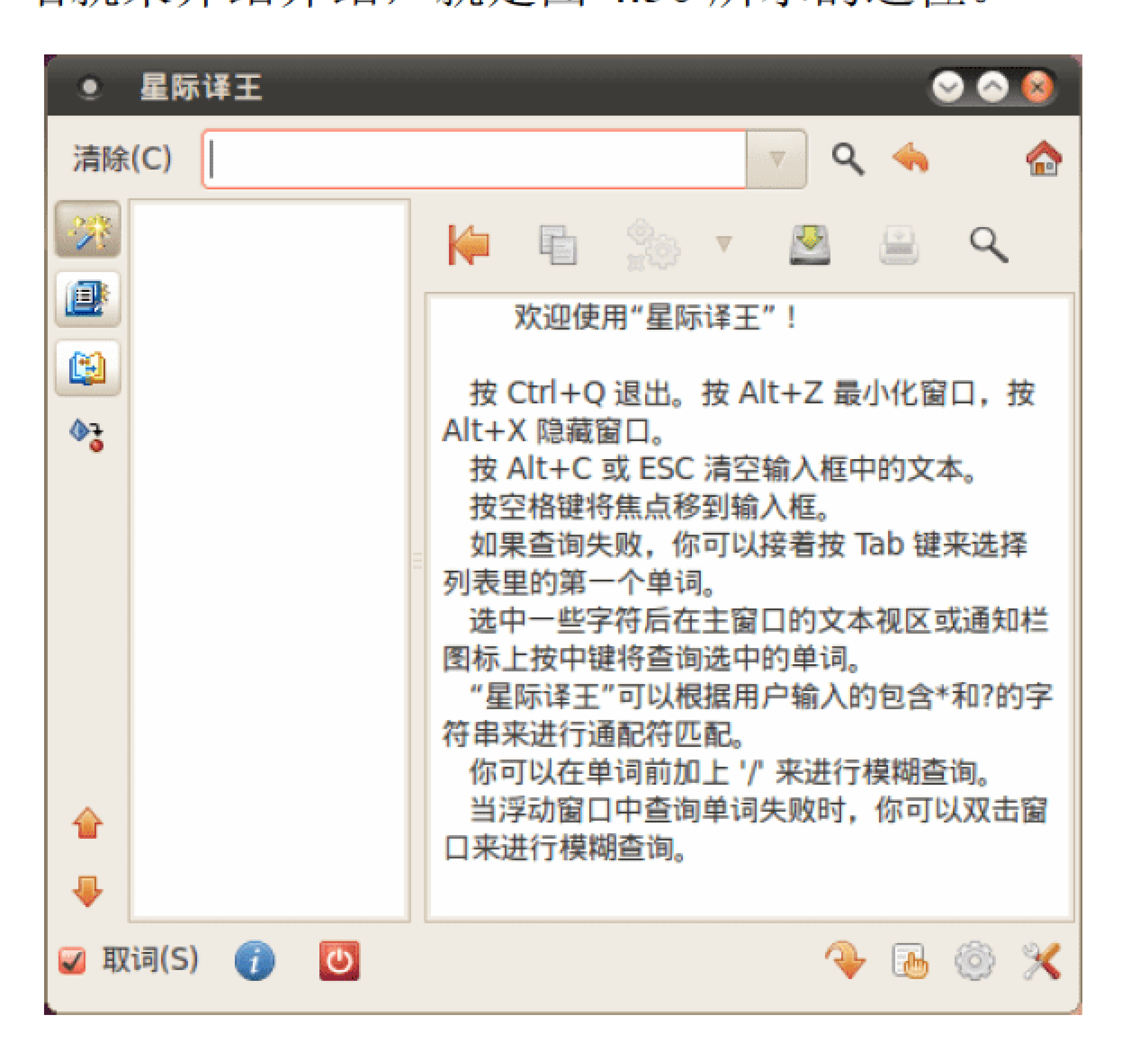


图 4.50　星际译王界面

星爷来自中国，是我们这里少数来自中国的软件之一。他可以说是我们这里最有学问的人，简直什么都知道。一开始他只是懂点英语而已，大家都只当他是个英语翻译，后来懒蜗牛叫狐狸妹妹 Firefox 去给他找来各种各样的书给星爷看——就是那种叫做字典的书，这种书只有星爷能看懂。看了这些书以后，星爷知道的就多了，什么日语、法语、德语都会了，估计联合国要开会请他一个人去当翻译就行了。于是星爷从英语翻译一下子晋升为

地球语翻译！（地球上的语言都会-_-）不知道他以后会不会再学学火星语？

提示：星际译王英文名称为 stardict，可以直接从软件源中安装。

然而星爷是不仅仅满足于当地球语翻译的，或者说，懒蜗牛是不满足于让星爷只做个翻译的。这之后他又给星爷找来了本《陈义孝佛学常见词汇》，于是星爷开始研究佛学了，不过这东西只是懒蜗牛他一时的好奇而已，后来就再没给星爷找过佛学的书，而是找了本《五笔 98》，开始学习五笔了。要说这输入法的事，那可是 ibus 的本行啊，可是无奈 ibus 老弟干活还行，表达能力不强。懒蜗牛要打什么字，ibus 可能很快反应过来，打在屏幕上，可是懒蜗牛要是忘了某个字怎么打，问问 ibus，那可要了命了，打死他也说不清楚啊，只能问："是 a 开头不是？"然后 ibus 把所有以 a 开头的显示出来数一遍，摇摇头："不是。"然后懒蜗牛再猜："是 s 打头？"ibus 再把 s 打头的列一遍……这样实在太费事，于是懒蜗牛就让星爷学五笔，到时候就能去问星爷："这个……'我'字用五笔怎么打啊？"哎，他也真好意思。

提示：屏幕取词方面，星际译王不支持鼠标指向取词，仅支持鼠标选中取词。取词效果如图 4.51 所示。



图 4.51　星际译王使用鼠标选中取词

会了五笔还不算完，后来星爷又看起了《本草纲目》，研究起了中医。不过他还不能当大夫，针灸号脉啥的他不会（就算会也没法号啊），那会啥呢？看了《本草》当然最精通的就是药理了，随便说一种药，他就能告诉你这个药的药性、药效、怎么用，等等。这时候基本上我们已经对星爷的全能感到震惊了。后来他又开始研究古汉语，看古汉语词典，康熙字典，一天到晚跟我们之乎者也："夫内核者，老大也，发其命，出其令，而统'康皮右特儿'……"

提示：可以到这里下载星际译王的词典：http://stardict.sourceforge.net。

我也趁没人的时候，问过星爷：您怎么懂这么多呢？看什么会什么。星爷很神秘地笑笑告诉我，其实他没什么本事，就是在学校学过信息检索而已。用户给了那么多本书，要想都记住，根本不可能嘛，他只是每次在用户问起事情的时候，赶快去查书，用最快的速

度找到并告诉用户。要是没了这些书，他知道的就少很多了。不过也不至于离开了书就什么都不知道，现在的星爷学会上网了，可以连接到一种叫网络辞典的地方，自己不会可以去网上查，不过那样效率自然不如自己翻书快了。

4.5.3　电子邮件 Evolution 和雷鸟

简历不是存起来看着玩的，得发出去给别人看才行。这不，懒蜗牛同学正在用“心有灵犀”（以下“心有灵犀”特指 EmPathy）登录他的 MSN 账号，然后给别人传简历——呃……这个方法是不是不大正规啊，哪有公司这么收简历的。

我问心有灵犀：“用户在给哪个公司发简历啊？”心有灵犀说：“咳，哪有公司这么收简历的，他是在给前几天来的那个 MM 发呢。”……怪不得呢，原来懒蜗牛只是在交流写简历的经验啊。我说：“估计是 MM 不会写，要拿他的来看看。”心有灵犀却并不关注用户的目的：“我虽然不知道为什么懒蜗牛给她发，但是我知道，那位 MM 肯定看不了懒蜗牛写的这个简历。”我正要问为什么，这时候，有一个人在旁边冷不丁地说了一句：“哼，干吗不用邮件传呢。”我一看，是 Evolution，我们这里的信使，默认的电子邮件软件。

【怀才不遇的 Evolution】

说起电子邮件，大概您不会陌生。不过要说邮件客户端，可能在公司上班的还会经常用，在家里用的人就很少了。一般上网的人虽然都会有自己的邮箱，但大都是通过网页来收发邮件的，用邮件客户端的人比较少。而我们 Ubuntu 又很少会用于工作环境（用也基本上都是做嵌入式开发的人），于是，Evolution 就成了我们这里最怀才不遇的人。EvoLution 运行界面如图 4.52 所示。

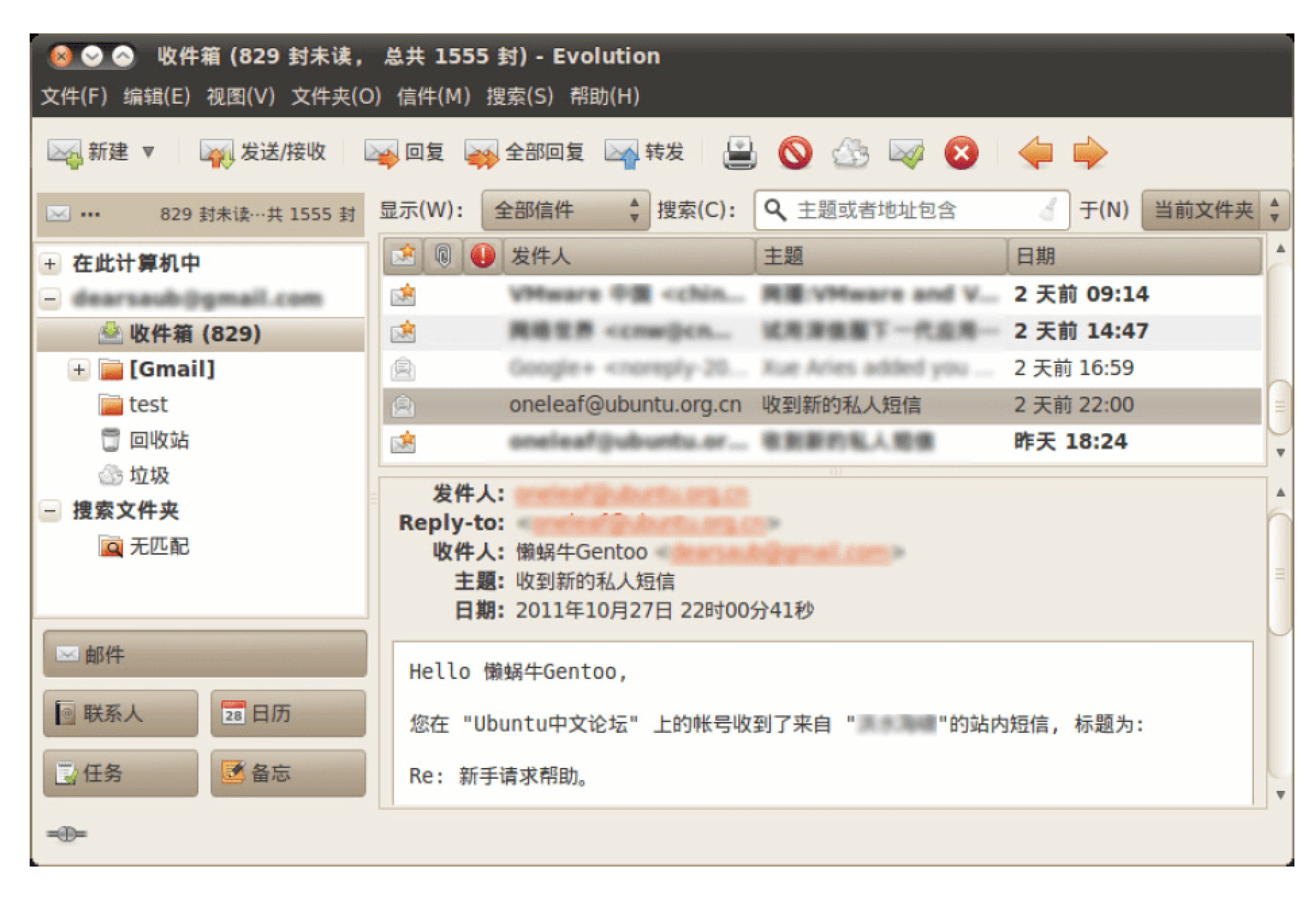


图 4.52　Evolution 运行界面

其实 Evolution 的本领还是可以的，他的工作基本上就相当于 Windows 7 那里的 OutLook，除了收发邮件，还可以安排日程。阅读邮件的时候，Evolution 也基本上什么样

子的邮件都可以正常显示，中文英文的都没问题。不过还是有很多狐狸妹妹 Firefox 的粉丝们不使用 Evolution，而是安装了狐狸的同门师姐——雷鸟姐姐，ThunderBird，也就是图 4.53 中所示的这个软件。

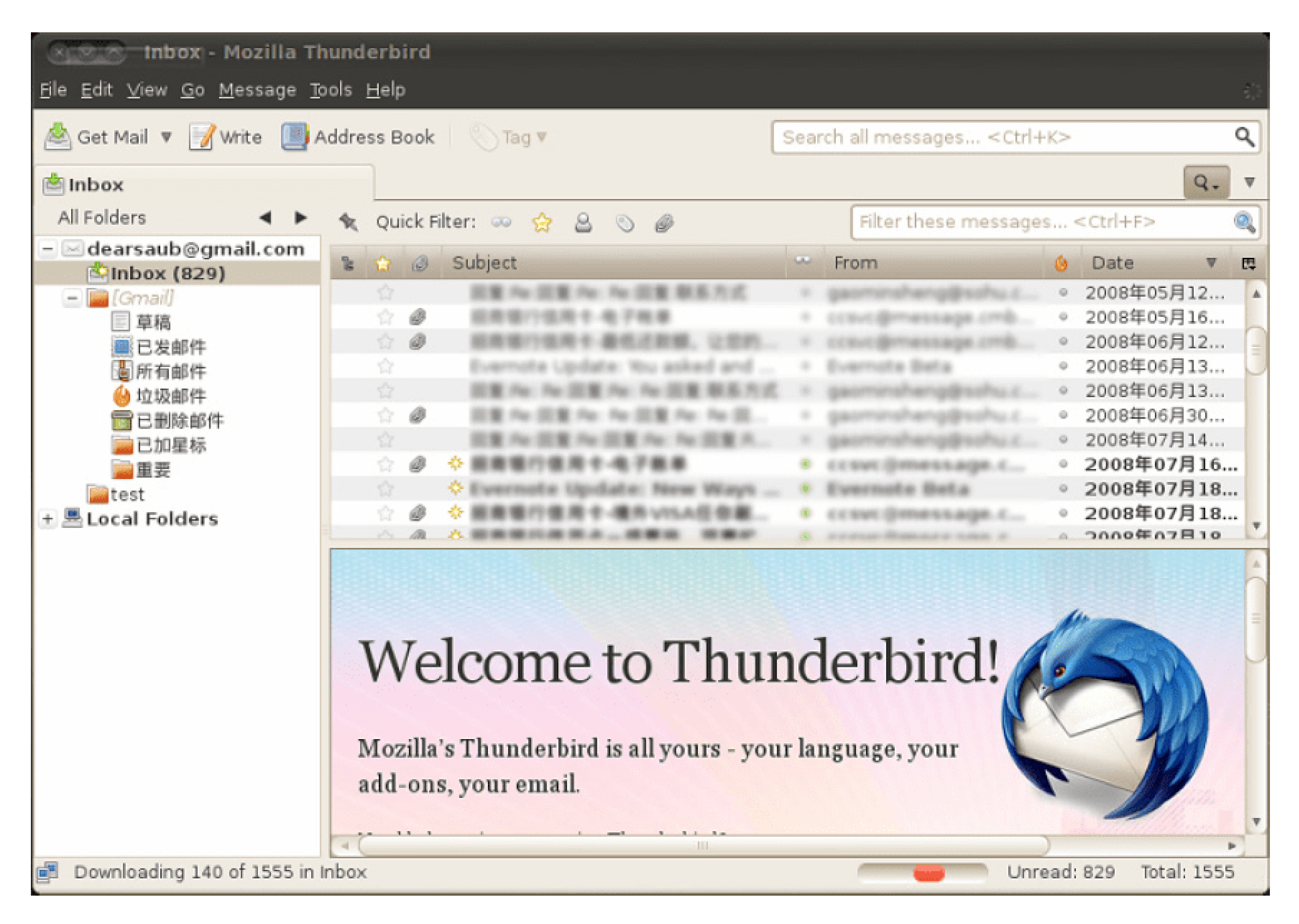


图 4.53　ThunderBird 运行界面

【雷鸟】

雷鸟姐姐和狐狸妹妹有很多共同之处。比如师出同门，都是 Mozilla 的得意门人；比如都可以安装丰富多彩的插件；比如都因为在 Windows 平台下表现良好而拥有很高的知名度。所以很多人慕名而来使用了雷鸟作为邮件客户端。

雷鸟姐姐能力确实不在 Evolution 之下，邮件的收发、显示自不必说。关键她长得也漂亮，而且简洁高效，符合 Linux 的哲学。可 Evolution 也有他独到的地方，那就是和 Exchange 的配合。很多公司的邮件服务器都使用了微软公司的 Exchange Server 作为邮件系统。除了他们自家的 OutLook 外，像雷鸟姐姐这些开源的客户端多数都不能够正常地连接到 Exchange 上，然而 Evolution 除外。经过多年修炼，Evolution 顿悟了 Exchange 的链接原理，他默认情况下就可以正常地与 Exchange 进行连接，收发邮件，这也是为什么他成为 Gnome 里默认的邮件软件的原因。

提示：最新版本的 Ubuntu 系统，已经改用 ThunderBird 作为默认的邮件客户端。

说了这么多，可惜懒蜗牛同学只是一个普通的家庭用户，并不需要一个邮件客户端，所以 Evolution 在我这里从来就没运行过，也难怪他没事就发牢骚呢。

4.5.4　与 Windows 的文档交互

正这时候，心有灵犀笑着对我说："你看怎么样，我就说这文件发过去也看不了吧。"我一看，果然，那 MM 正跟懒蜗牛抱怨呢："你发的是什么文件啊，打不开。"我明白了，懒蜗牛用 OO 老先生写完了简历，顺手就保存成了 OO 老先生默认的 odt 格式，这虽然是

个开放的格式，可是 Windows 7 那里的 Office 只有新版本的才能打开，老一点的版本，像 Office 2003 及之前的版本，都不能打开 odt 格式的文件。

🔔提示：Office 2007 SP2 以上版本可以支持 odt 格式。

估计 MM 那里的 Office 就是不支持 odt 的老版本 Office，所以无法打开。要我说这就是那个 Office 装蒜，就是诚心不愿意支持这个 odt 格式。你想，一个开放的格式，又不需要破解，又没有什么专利权问题，要想支持早就能支持了，何必非得等到快 2012 了才来支持这格式。不过也没关系，OO 老先生也可以把文件存储为 DOC 格式的，那你总不能再打不开了吧。懒蜗牛同学很快也意识到了这一点，于是把那个文档重新打开，另存为.doc 格式，再次发给 MM。MM 这回倒是看到了，可是问题又来了："怎么格式有点乱呢？"

唉，要说这实在不能怪 OO 老先生了，这个 DOC 格式是微软公司自己设计的，并且还不开放，别人是不知道这个 DOC 格式的文件应该怎么写的。可是 OO 老先生经过多年的潜心研究，分析过很多 DOC 文件，总算是研究出了这种格式的大致写法。可是毕竟只是研究出了一部分而已，如果文档里使用了一些复杂的结构，OO 老先生可能就搞不定了。所以文档的格式和字体的现实效果，与 Windows 7/XP 下的 Office 里看见的有些出入，也是可以理解的。可是我理解没用，MM 不理解啊。就算 MM 理解了，这简历发到人家公司的人事部去，人家一看这格式乱七八糟的，人家也不理解啊。这怎么办呢？说来也没啥好办法，只好用 Windows 7 的 Office 去写，或者把 odt 导出为 PDF 格式再发。OO 老先生要将文件输出成 PDF 还是非常方便的。

🔔提示：OpenOffice 可以方便地导出 PDF，只要单击"文件"菜单，选择"输出成 PDF"即可，如图 4.54 所示。



图 4.54　用 OpenOffice 输出 PDF

懒蜗牛同学最终还是让 OO 老先生把简历输出为 PDF 格式发送给了 MM，这也算是目前 Linux 和 Windows 平台交换文档时比较通用的一种方式了。MM 终于看到了懒蜗牛的格式规范的简历，并且通过心有灵犀，发过来一句评语："乱七八糟！"呵呵，看来就像 OO

老先生说的，懒蜗牛要重写了。

4.5.5 其他的办公软件

【国产的永中 Office】

那么有没有能更好地理解 DOC 格式的办公软件呢。还是有的，永中 Office 就是其中之一。

永中 Office 是一个 Java 程序，像星爷一样来自中国。他的目的就是成为能够替代 MS Office 的一体化办公软件。他对 DOC 格式文件的研究确实比 OO 老先生深刻和独到。基本上可以做到在永中下和在 MS Office 下看到的效果是一致的（当然偶尔也会有例外），图 4.55 所示就是永中 Office 的运行界面。然而可能由于他是 Java 的，所以运行起来效率不是很高。您听这名字，永中，听着就觉得臃肿。

再有，他也不开源，是一个闭源的软件，而且不支持 OO 老先生的 odt 文件格式，所以家庭 Linux 用户使用他的人不是很多。如果你想体验一下也很方便，只要去永中的官方网站下载就可以 http://www.yozosoft.com/download/zmo.jsp。

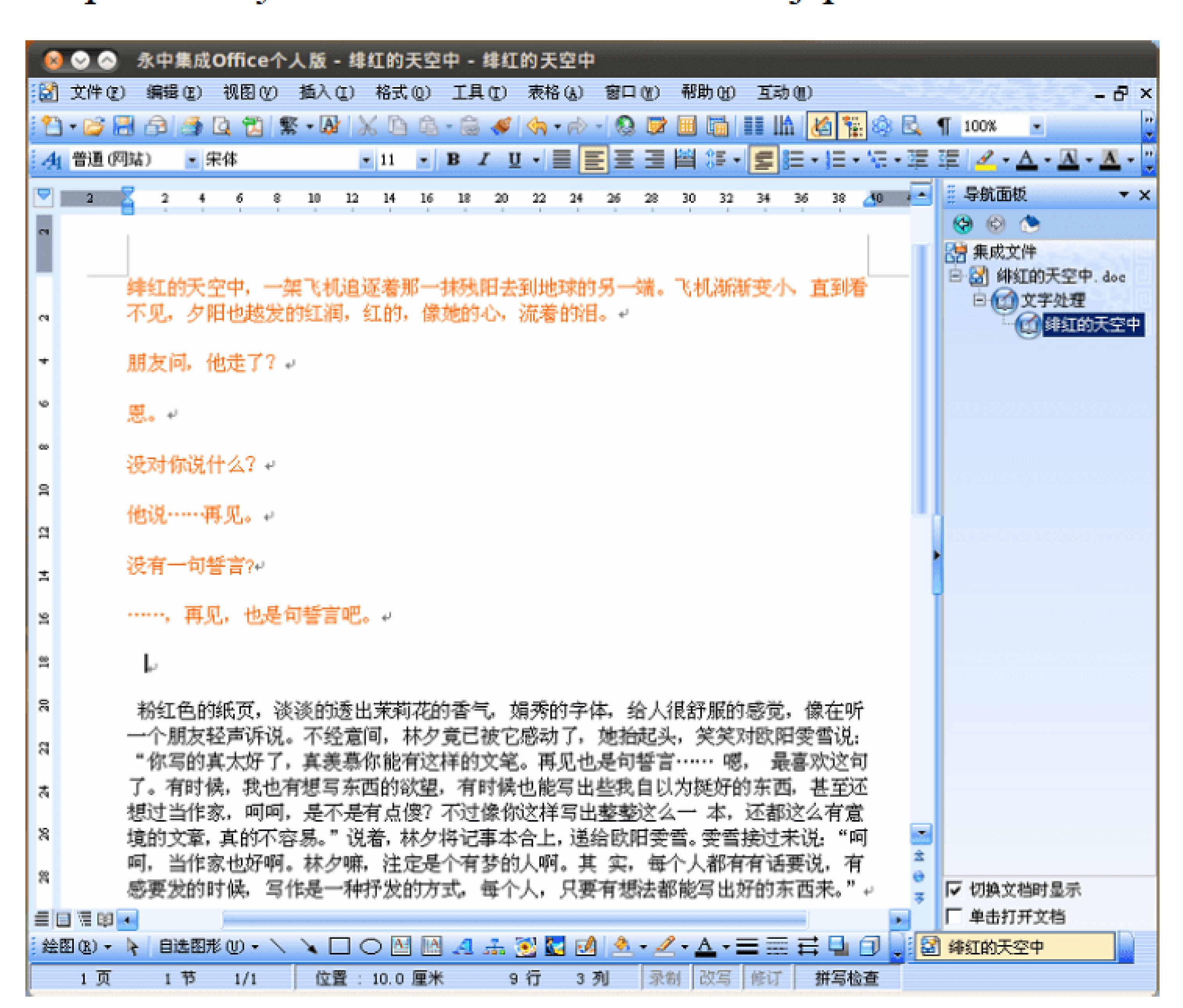


图 4.55 永中 Office 运行界面

【KOffice】

除了 OO 老先生和永中 Office 外，我们 Linux 世界里还有很多类似的办公软件。比如 K 派的 KOffice，就是比较有名的一个，图 4.56 所示就是 Koffice 的运行界面。虽然很多以 KDE 为主要图形界面的发行版依然会用 OpenOffice 来作为默认的办公软件，但其实 KDE 是拥有自己的办公软件 KOffice 的。KOffice 的特点是拥有广泛的工具集，除了文字处理、电子表格、演示文稿和数据库系统之外，KOffice 还包含 3 个图形处理工具——流程图应用、矢量绘图软件和光栅设计应用。还有项目管理工具、报表开发工具、制图绘图工具和数学公式编辑器。但是 KOffice 对文档格式的支持，尤其是对微软格式的兼容性远不如 OpenOffice，更不用说永中 Office 了（KOffice 喊冤：我是 Linux 软件，干吗非得支持微软

的格式！）。所以 KOffice 使用并不广泛。

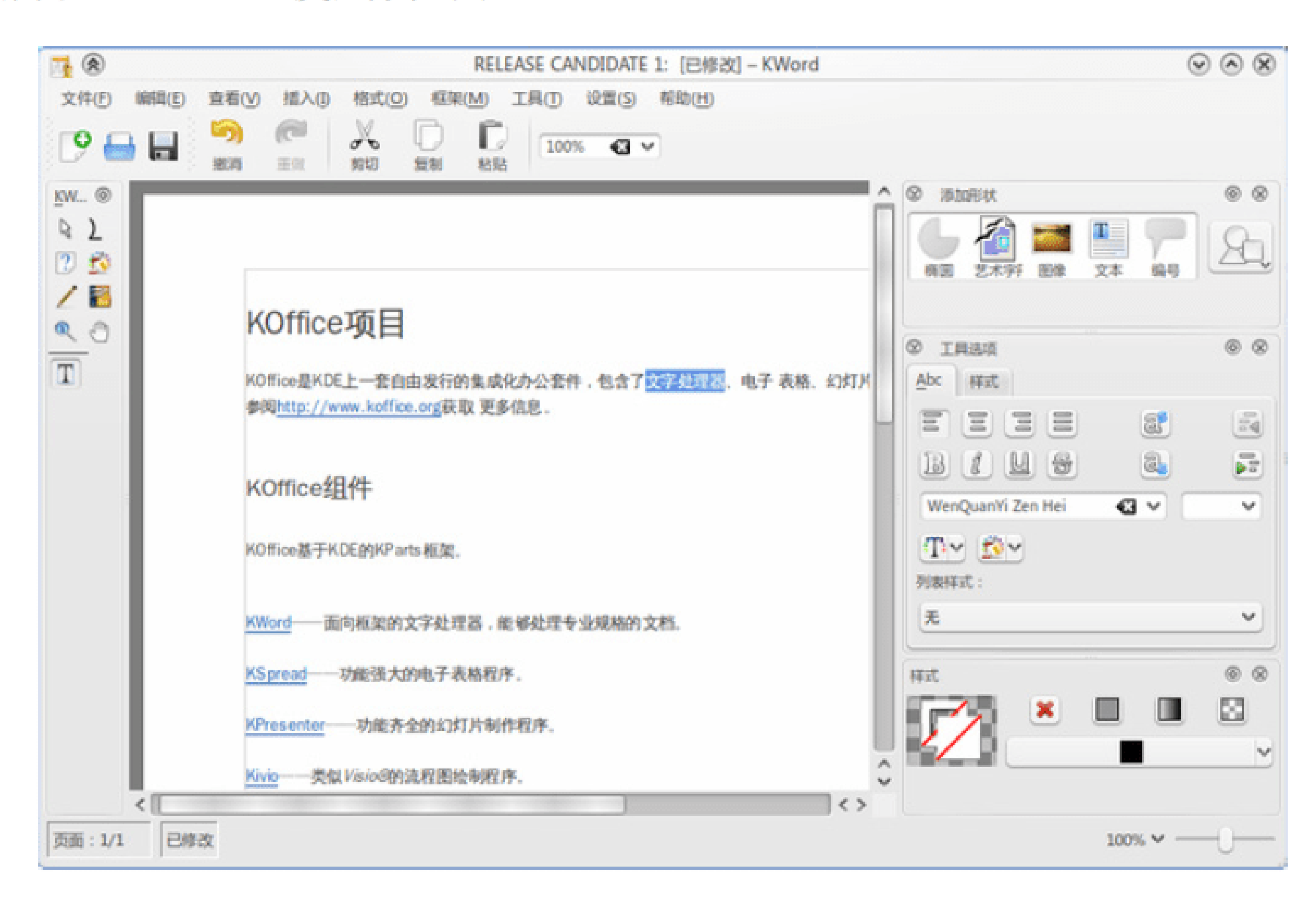


图 4.56　KOffice

【AbiWord】

如果你只是想有一个处理文档的小软件而不是包括了演示文档。电子表格等功能的办公软件，那么有一个小家伙你肯定喜欢——AbiWord，就是图 4.57 所示这位。听名字就知道了，他就是个 Word 替代软件，而不是什么 Office。这个小家伙只有十多兆大小，然而平常所使用的几乎所有的 Word 功能——信封、报表、邮件合并、修改跟踪等，他都可以做到。AbiWord 并不适合企业级专业人员使用，但是对于那些要求不是很严格的学生和个人来说，AbiWord 足够了。他支持 odt 和 DOC 格式，不过兼容性似乎同样不太好，而且打开包含多种语言的文档时会遇到点问题，但愿随着以后的发展会越来越完善吧。

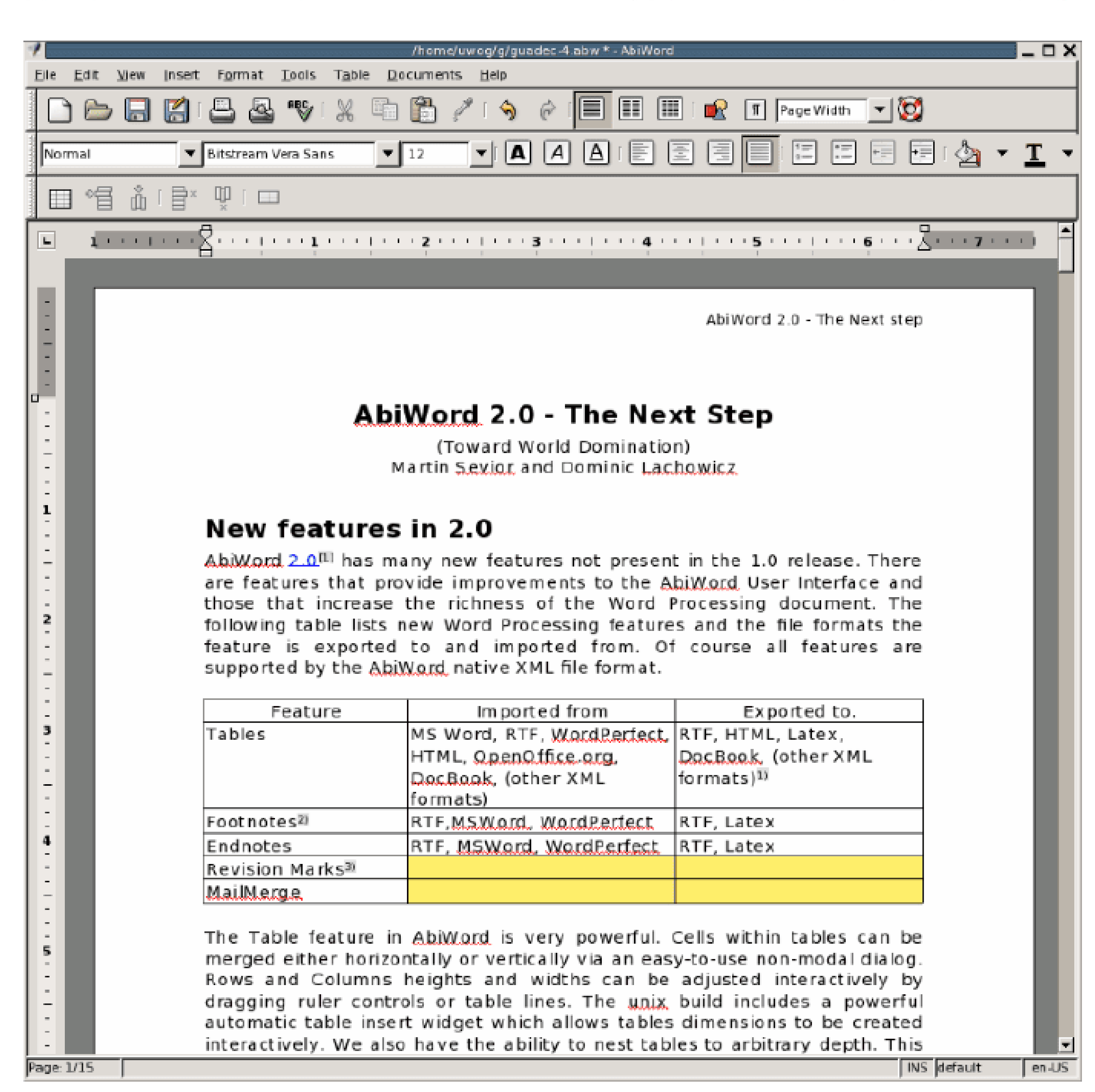


图 4.57　AbiWord

4.6 我的杀毒中心

虽然我们 Ubuntu 系统作为家庭用户的桌面版没啥中毒的机会，因此也没有装杀毒软件的必要，但是给我们装个杀毒软件，帮别的系统杀杀病毒还是不错的。这不，懒蜗牛同学就机缘巧合地开始研究 Ubuntu 系统下的杀毒软件了。

4.6.1 Linux 下也有杀毒软件

事情是这样的。懒蜗牛同学写好了简历后，把简历存储在他的 U 盘里拿去打印。然而这个学校的公共电脑，尤其是天天插各种 U 盘的连接打印机的那台电脑中，基本上快成病毒博览会了。懒蜗牛同学的 U 盘往那台机器上一插——无悬念命中。

懒蜗牛同学发现 U 盘染毒之后，马上想到了我们 Ubuntu 系统对病毒免疫（准确地说是对 Windows 病毒免疫）。于是打算把 U 盘插到我们系统上，备份好 U 盘里面的重要文档之后，把 U 盘整个格式化一遍。但转念又一想，这样也太没有技术含量了，不如在 Ubuntu 系统中装上杀毒软件，把 U 盘上的病毒杀掉。对，说干就干！

那么 Linux 系统下也有杀毒软件么？当然有，下面就给您介绍几个。

【Avira AntiVir Personal】

Avira AntiVir Personal 是一款来自德国的杀毒软件，俗称“小红伞”。这是因为他的东家——AVIRA 公司的 Logo 就是一把红色的雨伞，如图 4.58 所示。



图 4.58　AVIRA 的 Logo

这个杀毒软件的特点就是实时监控时的系统占用率低，节约系统资源。不过他的图形界面却是用 Java 实现的，反应可能会慢些。反正作为杀毒软件也不需要经常开着图形界面，所以影响也不大。另外，这是一个商业软件，但是对个人用户是免费的。如果想使用他可

以去他的网站下载：

```
http://www.avira.com/zh-cn/avira-free-antivirus
```

【Free avast】

这是一个来自捷克的免费杀毒软件。他非常小巧，没有常驻内存的实时监控病毒的功能，只能够用来扫描并清除硬盘、U 盘等存储设备中的病毒。他也是一款商业软件，同时，针对个人用户的家庭版是免费的，但是需要每年通过电子邮件注册一下，以获得新的证书。

这款杀毒软件也有图形界面，但是相当简陋，如图 4.59 所示。（然而有趣的是，这款杀毒软件的 Windows 版本以界面精美而著称）。

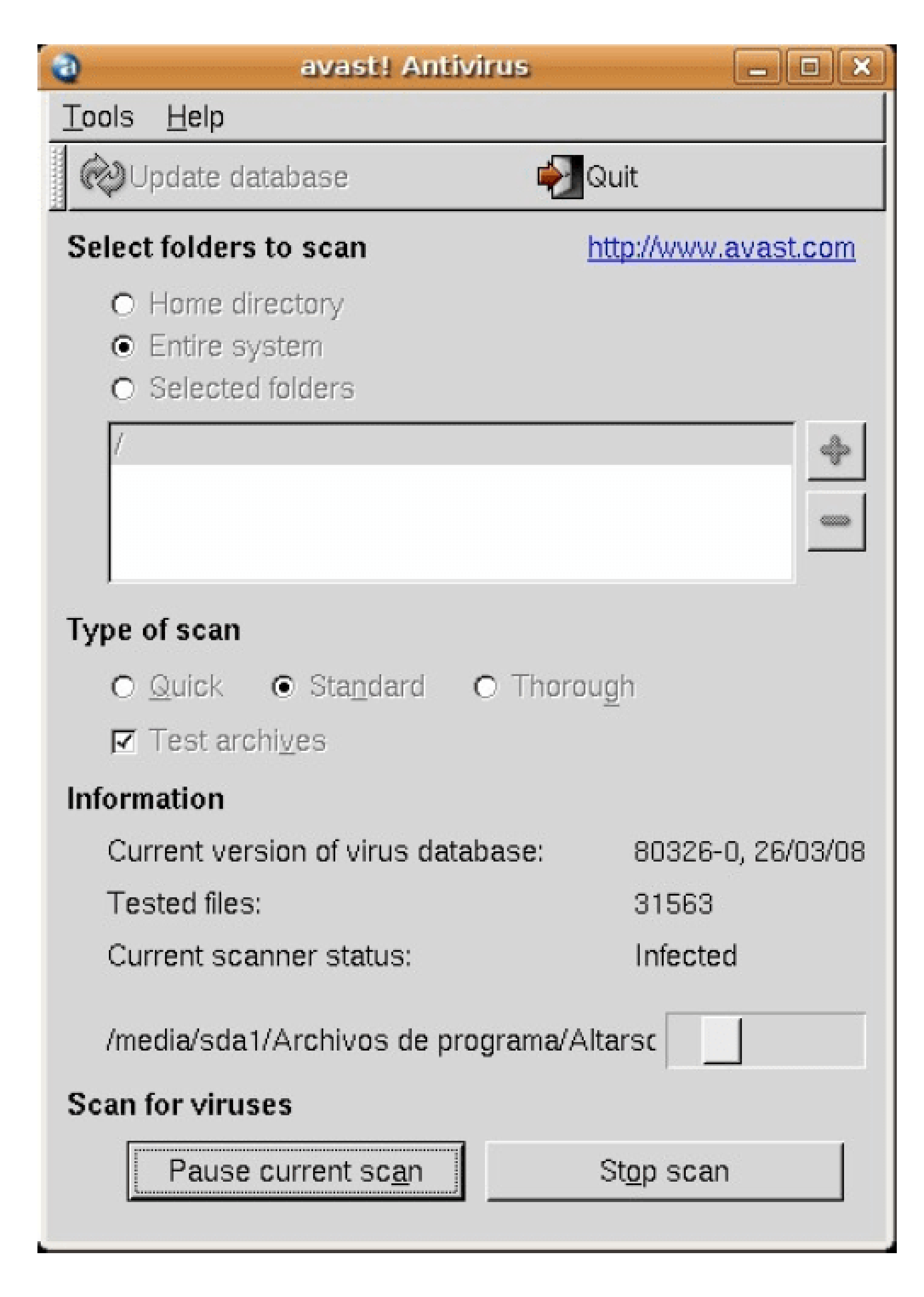


图 4.59　avast 界面

如果想安装这个杀毒软，还是去他的官方网站，有专门的 deb 包下载，方便安装：

```
http://www.avast.com/linux-home-edition
```

【ESET NOD32 Antivirus for Linux Desktop】

NOD32 这个杀毒软件在 Windows 系统中也是比较有名气的，在 Linux 系统下依旧表现不俗。无论是对系统内存中的病毒侦测、防御，还是对硬盘中病毒的扫描和查杀，做得都非常到位，界面也很漂亮，图 4.60 就是他在 Linux 系统中的图形界面。

然而各方面都优秀的软件是有代价的——他不是一个免费软件，需要付费购买。不过你也可以去官方网站下载下来体验一下，因为他有一个月的免费试用期。下载地址如下：

```
http://www.eset.com/download/home/
```

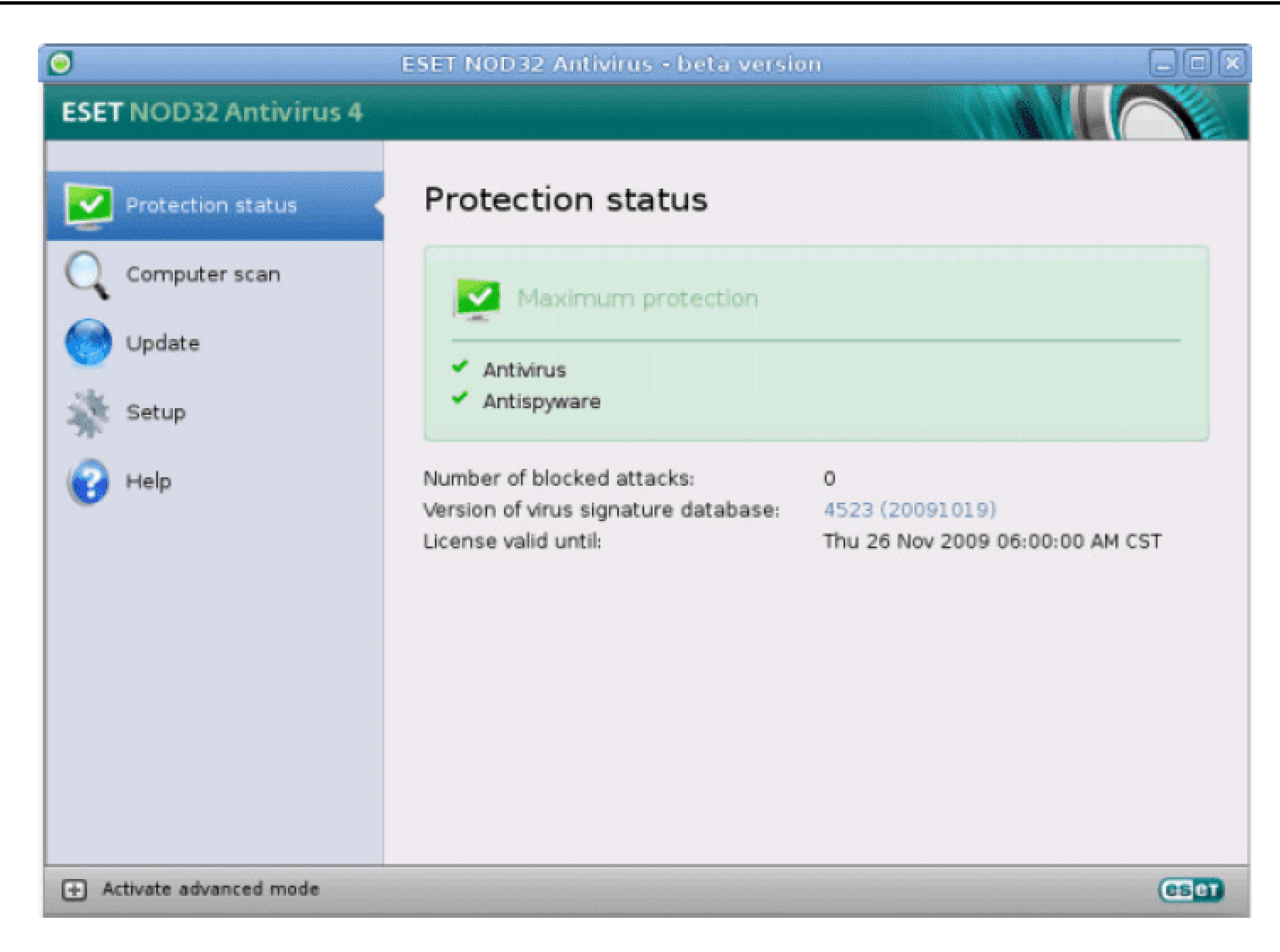


图 4.60　NOD32 运行界面

4.6.2　Linux 下杀毒毫无压力

懒蜗牛同学经过筛选，决定安装 avast 来完成他的 U 盘杀毒工作。于是，我们几个软件马上开始干活：狐狸妹妹 Firefox 掏出 Downthemall 扩展把 avast 的 deb 格式的安装包下载下来；超级牛力负责将安装包解开，掏出里面的 avast 并把他安顿在硬盘里；之后，我接到了蜗牛用户的指令，把 avast 叫起来干活。

avast 进入工作状态之后，先去网上下载了最新的病毒库。这个病毒库就相当于一沓子通缉令。那上面写着各种已知病毒的名字、相貌特征、作案手法等信息，以便 avast 在杀毒的时候查对。下载完毕之后，就见 avast 收拾好工具，整理一下装备，向着懒蜗牛那狭小的 U 盘出发了。

【活着的病毒不好杀】

要说起病毒来，Windows 系统还真是挺害怕它的。无论是 Windows 7 还是 Windows XP，或者更早的 Windows 系统，都需要杀毒软件的保护。针对 Windows 系统的病毒多种多样，各有各的本领，真是八仙过海各显其能。

有的病毒会伪装成别的软件。比如 Windows 7 叫醒“记事本”去干活，却不知真正的记事本已经被病毒一棍子打死了。现在躺在那里，长得跟记事本一样的家伙，其实是整了容的病毒。有的病毒能够藏在正常的程序里面。一个正在工作的 IE 同志，很可能工作服的兜里就隐藏着病毒。并且现在多数的病毒都会随着 Windows 系统一起起床。当 Windows 7 被叫醒，伸个懒腰揉着眼睛走进内存的时候，他庞大的身躯后面可能正趴着 40 多只病毒。

由于病毒是活的，要杀掉它们很困难。它们可能会有很多人共同作战：杀毒软件杀掉了内存里的强夫，内存里的大熊会把硬盘里强夫的复制版再叫起来。扭头杀毒软件去杀大熊，强夫会把杀掉的大熊抢救过来，结果谁也没被杀死。有的病毒更暴力，自己先跑进内存，一看见有杀毒软件要进来，立刻过去一铁锹把杀毒软件拍死，然后藏起铁锹装着杀毒软件的声音说：“杀毒软件成功启动，没有发现病毒，噢耶！”有的病毒还能监视 IE，一旦他要访问什么杀毒防毒相关的网站，二话不说，直接将 IE 干掉！

综上所述，要想在 Windows 系统中杀光这些活着的病毒，还是有很大难度的。

【睡着的病毒不反抗】

但这回 avast 去杀毒就简单多了。因为他杀毒的时候，电脑中运行的是我们 Ubuntu 系统。Windows 的那些病毒根本无法在我们的系统中运行，就像死掉一样只会躺在硬盘或者 U 盘里睡觉。这时它们不会有任何反抗能力。avast 过去，只要根据通缉令一一对照并干掉即可。只听到 U 盘里边不时传出“啊！”“呃……”“哎呦~”“我死得好惨呐！”等惨叫声。过了一段时间，avast 回来向蜗牛报告：“共发现病毒 7 种，总计 214 只，全部歼灭。”

从那以后，懒蜗牛又装了几款其他的杀毒软件，并经常帮他的同学和朋友清理各种移动设备上的病毒。谁有什么带毒的，不敢往自己电脑上插的设备，都要先插到蜗牛同学这台百毒不侵的电脑上处理一下，才敢使用。懒蜗牛同学成功地依靠我们 Ubuntu 系统，把他这台电脑打造成了一个杀毒中心！

4.7　本 章 小 结

这一回中，懒蜗牛同学算是过足了装软件的瘾了。上网用的浏览器挨个换了一个遍，下载软件、媒体播放软件、图片处理软件、办公软件、杀毒软件，都是挑了又挑，选了又选，终于都找到了自己用着顺手的一款。说明这 Linux 下的软件，还是挺丰富的吧。

那么这些软件能不能完全满足懒蜗牛的需要呢？会不会有什么软件在 Linux 下找不到替代品呢？咱们下回再说。

第 5 章　虚 虚 实 实

经过了工作间内激烈的角逐，各类软件的选拔工作已经尘埃落定。BT 下载类软件中，奔流脱颖而出；视频播放软件里，SMplayer 异军突起；图片管理软件组，Picasa 稳操胜券；网络浏览器呢，自然是 Firefox 成功卫冕。然而，虽然 Firefox 胜过了 Chrome 和 Opera 这样的对手，却不知道有更要命的兼容性问题在前面等着她，也等着我们的用户，懒蜗牛同学。

5.1　红酒大师 Wine

最先遇到的兼容性问题，就是浏览器的问题。当遇到一个狐狸妹妹（以下“狐狸妹妹”特指 Firefox）、Opera 姐姐，以及 Chrome 都束手无策的网站时，该怎么办呢？

5.1.1　非 IE 不可的网站

这一天懒蜗牛正带着狐狸妹妹徜徉在网络的海洋中，忽然想起一件事情——该发工资了。

原来，前一阵子他写的简历还真起到了些作用，目前正在一家小公司工作。3 个月的试用期刚刚过去 1 个月，懒蜗牛就惦记起自己的第一笔工资了。但是他的工资卡的发卡行——××银行的网点很少，最近的一个也要坐 8 站汽车走 3 里路才到，于是懒蜗牛决定让狐狸妹妹上网查查得了。

【遭遇 ActiveX】

说时迟，那时快，转眼间狐狸已经跑到了网上银行。懒蜗牛向上望去，只见页面上赫然写着 4 个大字，分上下两行。上面一行写着：卡号，下面一行写着：密码。然而奇怪的是，账号和密码后面，都没有输入框，效果如图 5.1 所示。

懒蜗牛看了一时摸不着头脑，仔细一看，底下还有小字：“为了您的账户安全，请安装××网上银行提供的××安全插件。”懒蜗牛顿时醒悟：哦，这银行的网页自然安全系数不一般，要装安全插件才能访问。好，那就装吧。可是当他按照网页上提供的链接地址把这个插件下载下来后就傻眼了——怎么是个 EXE 文件呢。

图 5.1　需要 ActiveX 的登录界面

这个 EXE 文件是 Windows 系统下的可执行文件，这种软件只能在 Windows 下运行，在我们 Linux 系统下面可是无法工作的。首先语言就不通，我们 Linux 软件说的话他们 Windows 软件完全听不懂，反过

来也是一样；再有就是他们 EXE 软件运行时需要调用各种库文件，也就是那些 DLL 文件，像图 5.2 里那样的一大堆，我们这里也没有。我们这里的库文件都是.o 或者.so 格式的，如图 5.3 里所示。所以 EXE 在我们这里是没法直接运行的。懒蜗牛也很迷茫，不知道该怎么办。不过有句话说得好：内事不明问老婆，外事不明问 Google。

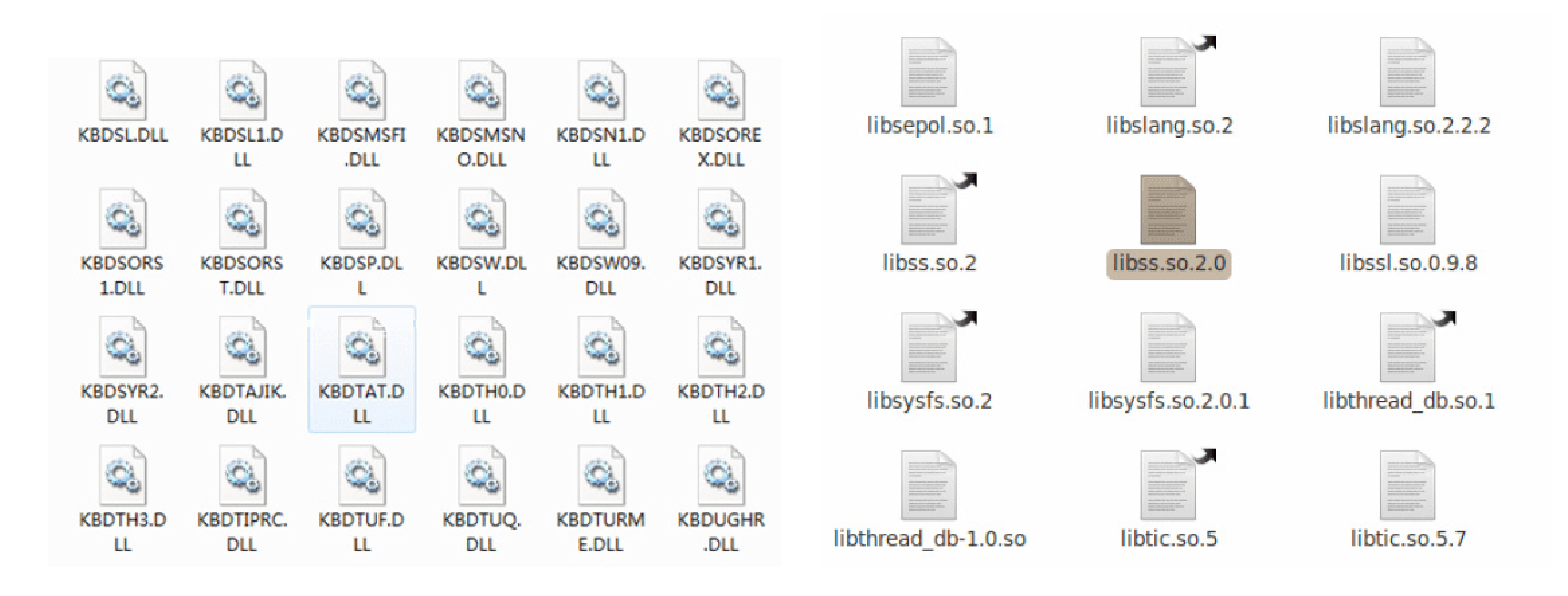


图 5.2　Windows 系统下的库文件　　　　图 5.3　Linux 系统下的库文件

懒蜗牛这一问才知道，原来，这些国内的银行大都使用了不大安全的 ActiveX 技术来写网银的安全控件。然而这 ActiveX 技术是微软公司发明的，人家是自产自销，只有自己家的 IE 浏览器能够支持这种技术，其他的浏览器想都不用想了。

提示：由于 ActiveX 的安全性风险，IE8 浏览器已经默认禁用一些 ActiveX 的特性。

狐狸妹妹的扩展和插件虽然不少，可是没有一个可以支持 ActiveX 的。只见狐狸急得直跺脚，可着急也没用，她也只能灰溜溜地回来汇报——我实在搞不定这个网站，你就是掐死我，我也没辙。

提示：Windows 版本的 Firefox 可以通过安装 IETab 扩展来支持使用 ActiveX 的网页。该扩展的本质就是临时调用 IE 来显示网页。因此 IETab 仅限于在 Windows 系统中使用。

懒蜗牛同学只好陷入了无奈之中，难道为了查看一下工资卡里的余额，竟然还要重启一下去一趟 Windows 么？不行，还得上网搜搜，不可能只有我一个人碰到这个问题吧。于是，他又牵过狐狸妹妹去网络上搜索了。他这边搜着，工作间里的软件们也没闲着，大家七嘴八舌地议论起来……

【狐狸的委屈】

“今儿个丢人丢大了！

“想我大名鼎鼎的火狐狸，什么样儿的网站没去过？什么样儿的网页没显示过？什么 Konqueror、Epiphany、lynx，除了那个自以为是的挪威妞儿 Opera 以外，哪个浏览器见了我不得叫声大姐？你说是在线视频还是 Flash 游戏，是 IPv6 还是数字证书，咱哪点儿含糊过？再说有我一身的插件和扩展，兵来将挡水来土掩，就算有什么功能我本身不支持的，也可以靠装插件和扩展来应付。可是今天，就今天，竟然就有网站我拿它没辙！

提示： Firefox 默认开启对 IPv6 的支持，如果你的网络不支持 IPv6，可以关闭它以提高响应速度。

在 Firefox 地址栏输入 about:config 并回车，在出现的页面中找到 network.dns.disableIPv6 键值，将其设为 true 即可，如图 5.4 所示。

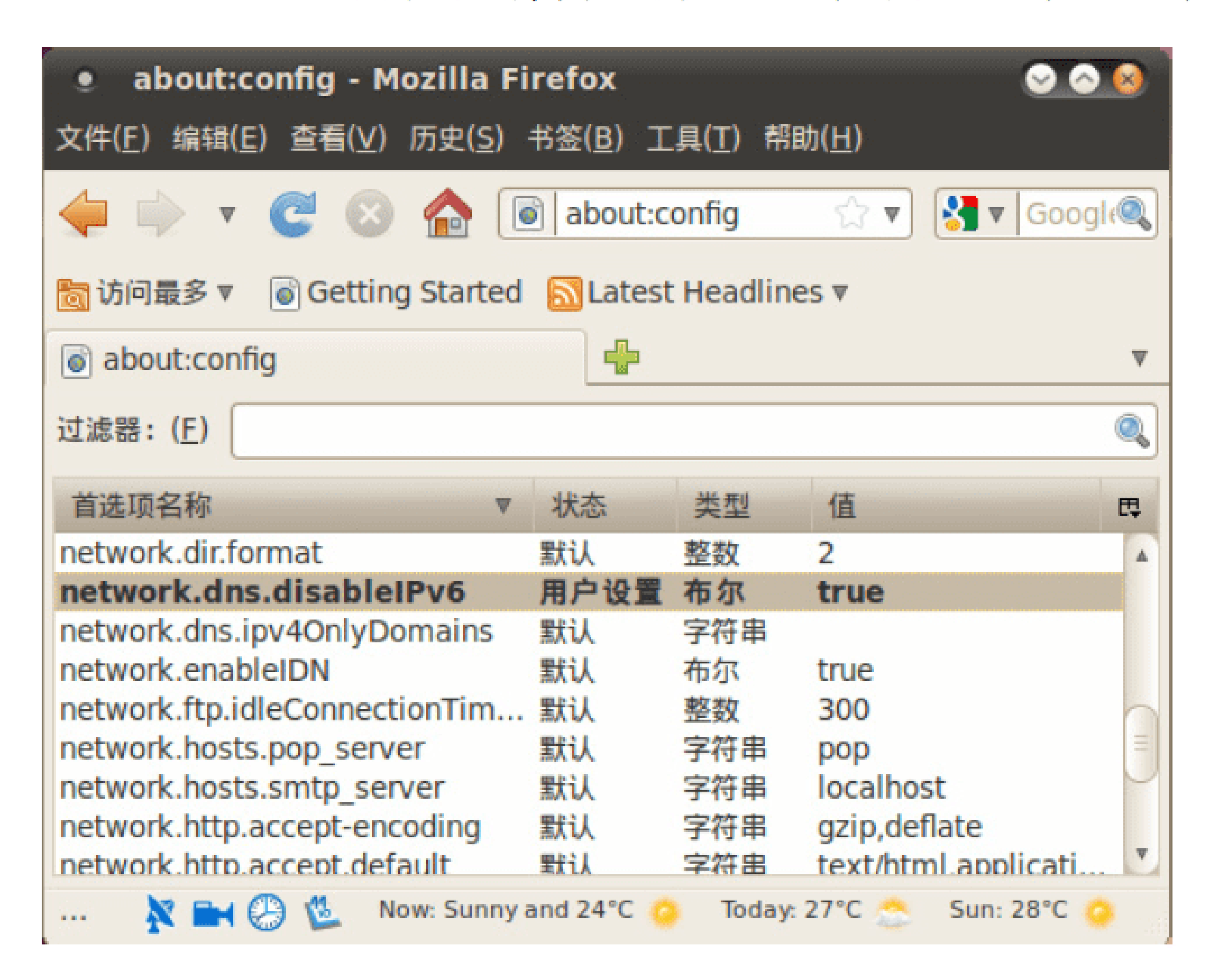


图 5.4　关闭 Firefox 的 IPv6 支持

“这破网站是一个银行网站，要说那好多国际知名的银行咱也见过，人家啥软键盘啊、证书啊、HTTPS 加密也都用得挺好的，也没见用什么 ActiveX，也挺安全的。怎么就这破网站非得用 ActiveX 呢？我其他的都会，就不会这 ActiveX，您说这不是存心揭我短儿么。

提示： 软键盘，指输入机密信息时网页上提供的键位不规则的虚拟键盘，如图 5.5 所示。通过软键盘来输入，避免了本地恶意软件劫持按键记录。

图 5.5　软键盘

“再说了，不会也不怪我啊，我倒是想会呢，谁教我啊。人家微软才不会把这个教给我呢，藏着掖着都来不及。结果可好，到了那网站，账号密码倒显示得挺大，可就是没有输入框儿，这不是白搭么！用户问我应该往哪输入，我哪儿知道啊！你说这不是让我在用户面前丢脸么。好在人家用户也理解这不是我的错，就算叫什么 Chrome、Opera 他们来也一样是白费，只有那倒霉的 IE 才行。可人家是微软的嫡系软件啊，怎么可能有我们 Linux 系统的版本。

提法：HTTPS 即 Hypertext Transfer Protocol over Secure Socket Layer，是一种通过加密实现的安全的 HTTP 连接。

“这不，用户没辙了，赶紧去上网查查到底有什么方法能够解决问题，我们这工作间里面也吵吵起来了。我们头儿让牛哥（也就是超级牛力啦）去查查，看有什么软件能够解决这个网络银行的问题。牛哥就掏出他的小本本一页一页地翻着。咱也别闲着啦，赶紧去网上看看有什么扩展能凑合解决问题吧。结果一查还真有，有个 wmlbrowser 扩展，就在我们官方网站下载，就是图 5.6 所示的这个。

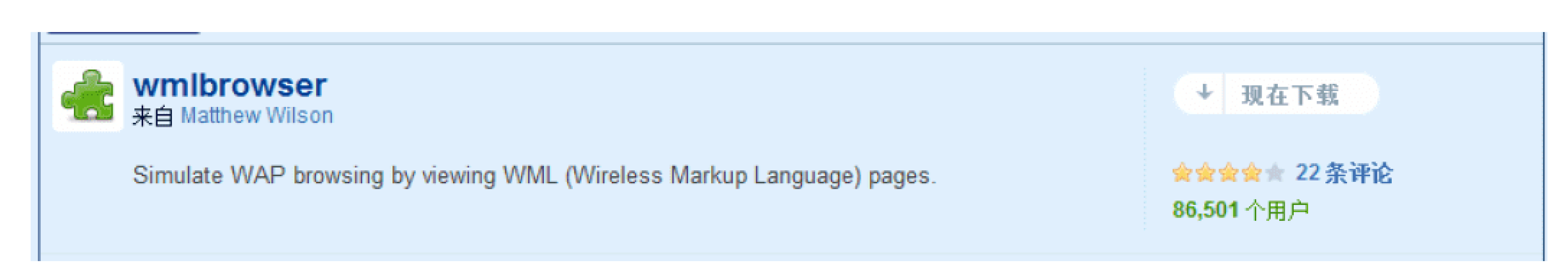


图 5.6　wmlbrowser 插件

“这个扩展本身跟银行没有什么关系，不过有了它我就可以支持 WAP 网页了，就是手机的那种网站。其实我本来也可以浏览一些 WAP 网页的，不过支持得不是很全面，有了这个扩展就好了。现在好多网络银行都可以用手机登录进行操作，手机上总没那个该死的 ActiveX 了吧。我就伪装成一部手机中的战斗机，去登录那个银行的 WAP 站点，这不就可以解决用户的问题了？我把这个发现赶紧告诉了我们头儿，他听了也挺美，可这时候窜过来一只 Chrome 说：‘不行的，我刚刚查了，那个××银行根本没有开通 WAP 网页。’嘿，这小子倒是真快。这回可好，刚有的一点希望稀里哗啦地就被浇灭了。

提示：WAP 即 Wireless Application Protocol，是一种向移动终端提供互联网内容和先进增值服务的全球统一的开放式协议标准，是简化了的无线 Internet 协议。

“这时候牛哥也没精打采地把他那小本本一合，无奈地说：‘唉，本 apt 有超级牛力，可也没有网银软件啊。’心有灵犀说：‘懒蜗牛应该去问问他的同事们，看他们都发了没发就知道了。’Chrome 反驳道：‘那也顶多是解一时之困，这网上银行的问题还是没有解决嘛。’这帮人也是瞎出主意，要登录银行网页就得通过我啊，我都没辙，你们能有什么辙？他们这一吵吵，结果给吵吵醒了一位，揉着眼睛问：‘What happened?’唉，毕大师这鸟语我们是听不懂啊，好在有毕翻译，赶紧问我们：‘怎么了？出什么事了？’牛哥过来解释：‘有个银行……’刚说到银行俩字，忽然牛哥眼里光芒四射，过去抱着毕翻译的肩膀不住地一边摇晃一边喊：‘对呀，就是你了！就是你了！’”

5.1.2　安装 Wine

【超级牛力的顿悟】

“本 apt 今天算是没超级牛力了，甭说牛力，羊力都没有了。工作间里上上下下里里外外，都让一个叫做××的银行网站给难住了。

“要说这个网页也好，ActiveX 也好，这些是狐狸的专业，我是不懂啊。我就知道，这 EXE 在我们 Linux 这一亩三分地上是运行不了的。你一个银行既然不装插件登录不了，可

又只提供了 EXE 格式的插件，那狐狸肯定没辙，除非那银行提供狐狸的插件。我也查遍了整个软件源，没找到跟银行，跟 ActiveX 有关的软件。

“正在一筹莫展呢，忽然看见毕大师和毕翻译出来，在这电光石火般的一瞬间，我的思维忽然像被苹果砸了的牛顿一样清醒透彻——毕大师，他不就是 Windows 的软件么，他不就正在我们 Linux 系统下干活么，他不就因为有个毕翻译么，这个 IE 也照样可以有！

“根据拥有超级牛力的本 apt 的资料来看，这个毕翻译，他出身好像是个卖酒的，可能主要是卖红酒吧，所以大家给他起名叫做 Wine，图 5.7 所示是他的头像。关于 Wine 的故事，有很多的传说，我就说说其中的一个吧！



图 5.7　Wine 的 Logo

“相传，Windows 世界和 Linux 世界向来泾渭分明，老死不相往来。两个世界的软件语言不同，理念也完全不一样，因此要想沟通极其困难。然而 Linux 世界的人们毕竟还是开放和包容一些，终于在 Linux 界出现了一位隐士，改变了这一切。

“他表面上是个卖酒的酒保，却不甘于终身伴杜康为伍，一世听觥筹交错。他用自己的精力去学习 Windows 世界的语言；了解 Windows 世界的文化；感受 Windows 世界的气息。最终学会了与 Windows 世界的软件交流的手段。于是，很多的 Windows 世界的软件被 Wine 带到了 Linux 这片天地，为 Linux 界带来了别样的色彩。不过，Wine 并不能把 Linux 界的软件带去 Windows 界，即便是他自己也无法过去，不知道这是不是因为 Windows 界太过排外的缘故。总之，Wine 最终成为 Linux 系统中，所有 Windows 软件可靠的翻译，通过 Wine，Windows 的软件才得以进入 Linux 的世界。

提示：目前并没有在 Windows 环境中模拟运行 Linux 程序的软件。这是由于 Linux 系统中的软件多数为开源软件，因此可以很容易地移植到 Windows 平台。例如 Firefox、OpenOffice、Pidgin 等 Linux 下的常用软件都有 Windows 版本。

“那么毕大师的翻译又是怎么回事呢？那是 Google 公司看中了 Wine 的这种特殊的能力。本来他们想创造出一个能够在 Linux 世界里崭露头角的 Picasa，但看到了 Wine 之后就改变了原来的想法，直接把 Windows 的 Picasa 叫来，再找到 Wine，让他们互相了解，磨合，取长补短，最终使 Wine 成为毕大师的专职翻译，也就是我们的毕翻译。

“当然，关于 Wine 的故事，都是听前辈们说的，我没有亲眼见过，也没有机会见。今天这一听说银行的事，我光顾着找跟银行有关的软件了，直到看见毕翻译才忽然想起来 Wine 这个传奇人物。既然有了 Wine 就能跟 Windows 的软件交流，那就能找来那个 IE 工作了吧？这不就把问题解决了么！我正要向我们头儿汇报，用户已经传来命令：sudo apt-get install wine。哈哈，果然是英雄所见略同啊！”

内存里边，大家正一筹莫展呢，懒蜗牛同学忽然发出命令，要超级牛力装一个叫做 Wine 的软件。我问超级牛力这个 Wine 是干什么的，超级牛力说，Wine 这个家伙正是可以让 IE 运行在我们的系统中的软件，还说他刚刚想到这一点还没来得及跟我说，用户就先知先觉地让他去装 Wine 了。估计这小子又是在吹牛，哪有那么巧的事情。不过既然 Wine 可以解决问题，一颗纠结的心总算是放下了不少。

🔔**提示：**软件源中的 Wine 可能不是最新版，如果想安装最新版，可以去这个地址下载 deb 包并安装：http://wine.budgetdedicated.com/archive/index.html。

5.1.3　模拟运行的 IE

转眼间 Wine 已经来到了我们的硬盘里，先是自我介绍了一下，说自己叫做红酒，承蒙朋友们抬举，都叫他红酒大师，不过自己也没什么太强的本事之类的客套话。之后问了问情况，我把目前的问题跟他说了，任务很简单，就是把 IE 叫醒去干活。红酒点点头，拎起自己的工具包就走进隔壁 Windows 7 的屋里去了。

【催眠大师】

只见红酒大师先掏出好几块屏风，在 IE 周围挡了个严实，他说要用高科技全息投影设备在屏风上投出 IE 平时工作的虚拟环境，让 IE 觉得自己还是在 Windows 7 的领导下干活。解释了半天，反正我们也不大懂。过了一会儿，听见他在屏风内，用低沉浑厚的声音向 IE 念着：“现在，计算机正在启动，我是 Windows，我是 Windows，你要开始工作，你要慢慢醒来，你要慢慢醒来开始工作。醒来……醒来……”

随着他一步一步的引诱，屏风里渐渐有了动静，好像是 IE 迷迷糊糊地起床了。又过了一会儿，只见屏风打开一块，IE 慢慢地向工作间走去。

他走路有点不稳，一边走，红酒还一边跟在他左右不停地引导：“你像每天一样起床，正走向内存里，开始你的工作……”然后扭头问我：“让他去哪里？”看得快入迷的我这才反应过来，还没交代清楚具体的任务呢，赶紧说：“哦，让他去那个××网上银行。”红酒继续对 IE 说：“现在……Windows 让你去 XX 银行的网站……去吧……去吧……像每天一样……”这个时候，基本上所有人都看呆了，没想到这 IE 竟然就这么被红酒指使着干活去了，这哥们儿还真是个大师啊，至少也算是催眠大师。

🔔**提示：**由于 Wine 是模拟 Windows 的运行环境，因此 Windows 软件在运行时效率会有明显下降。

看着 IE 干活的样子还真是很搞笑，估计是因为他不是在清醒状态，所以基本上跟梦游似的在那里干活。虽然没出什么错，可动作慢了不少，懒蜗牛也只能姑且忍受一下。我们还照了张 IE 在我们这里干活时的照片来留念，如图 5.8 所示。

狐狸妹妹在一边好像是想偷偷跟他学学 ActiveX，可是他嘴里念叨着叽哩咕噜的东西我们都听不懂，只有红酒大师能懂他的话。我要发送命令，只能先告诉红酒，再由红酒翻译给 IE。

IE 要用什么东西，什么 DLL 文件啊，配置文件啊，红酒都赶快给他找来，原版的找不来就自己做一个差不多的，放的位置都跟 IE 在 Windows 7 那里干活时的位置差不多，让 IE 以为自己还是在 Windows 7 那里干活。

等了好半天，IE 总算把事办完了，懒蜗牛赶紧把 IE 关了，还是叫过狐狸妹妹来去浏览其他的网页。也难怪，要是我，我也得关，看着 IE 晕晕乎乎干活我就难受，哪有狐狸看着顺眼。

提示：配置 Wine 并安装 IE 的过程相对繁琐，可以使用 IEs4Linux 脚本进行自动安装。项目地址：http://www.tatanka.com.br/ies4linux/page/Main_Page。下载 IEs4Linux 脚本后在终端运行该脚本，即出现如图 5.9 所示的界面。

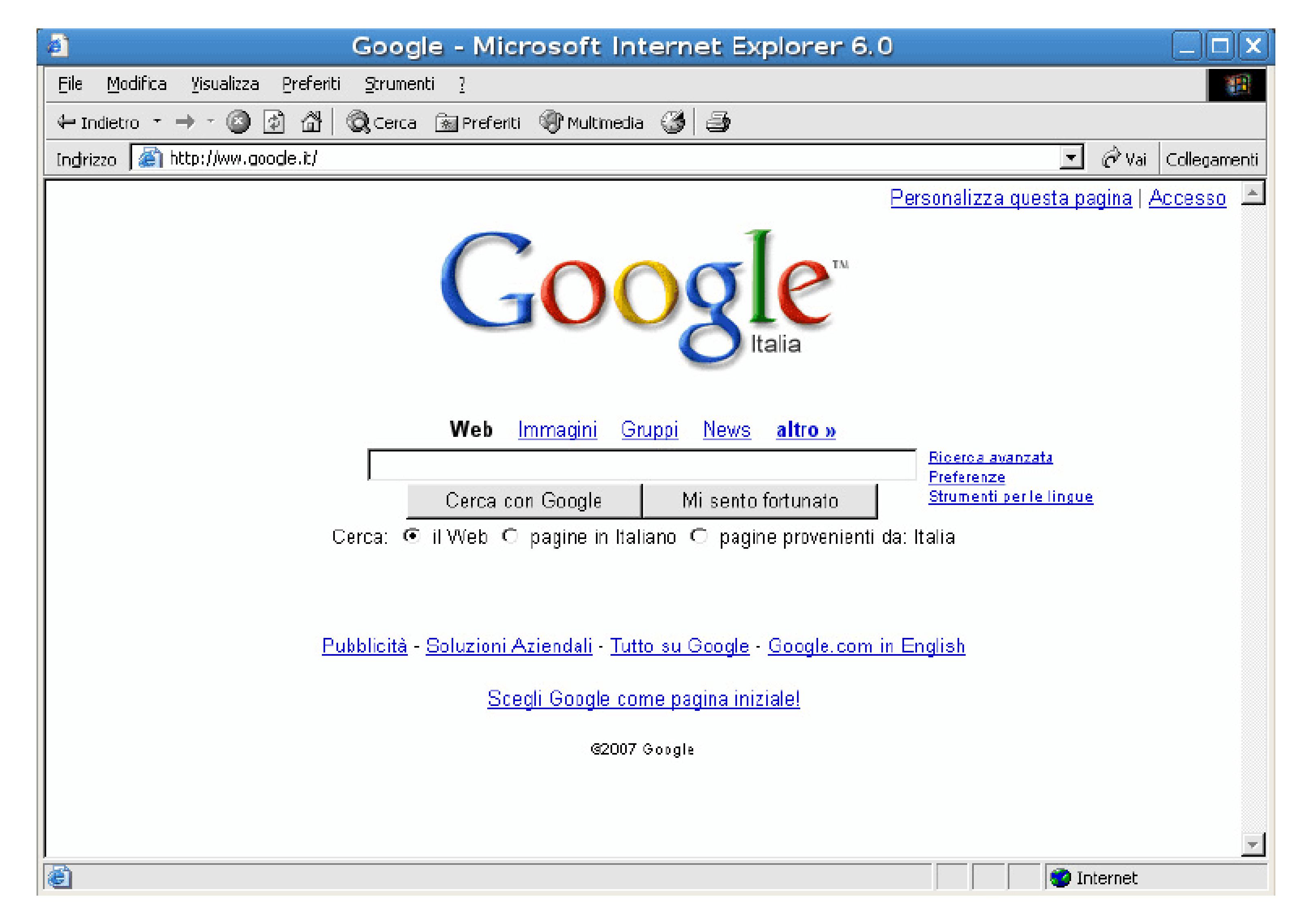


图 5.8 用 Wine 运行的 IE 6.0



图 5.9 IEs4Linux 运行界面

按照图 5.9 进行选择后，单击"确定"按钮即开始自动化安装及配置过程。运行 IEs4Linux 脚本前请确认已正确安装 Wine。

【IE 的困惑】

"唉，最近呀不知道怎么了，我这个脑子出问题了吧还是怎么回事，怎么总是昏昏沉沉的呢？而且好像记忆还不好了，要不就是有幻觉。我隐约地记得昨天明明起床干活了，好像是去了个银行，不过记得不怎么清楚，模模糊糊的感觉，好像做梦一样。

"我以前可不是这样的啊！我可是名门之后，血统纯正，我祖上从来也没有失忆的毛病。想当年啊，我们的祖先 IE5，就是如图 5.10 所示的这位。他那时候就跟着 Windows 98 混。那时候有个家伙叫 Netscape，可能有人见过，图 5.11 所示的是他的头像。这家伙觉得自己挺牛，基本上那时候上网的都得用它。可是，哼哼，有本事不如有靠山！我前辈 IE5 老先生虽然论本事……不比那 NetScape 强，可是他聪明啊，死粘着 Windows 98 老大，有了这强大的后台，慢慢地大家都开始用 IE5 了，NetScape 从此就销声匿迹。后来的 IE6 也是如法炮制啊，粘着 Windows XP，后来 IE7 横空出世了，就取代了 IE6 的位置，也跟着 XP 干……咳，我怎么说起这些了，看来脑子真是不行了啊。



图 5.10 Internet Explorer 5



图 5.11 Netscape 的 logo

“我明明记得昨天去了银行的网站，还是 Windows 7 叫我去的，可是今天我问他，昨天你说话声音怎么有点不对劲呢？是不是感冒了？他斜眼看着我问：‘昨天？哪有活啊？’我说不对呀，昨天不是你让我去那个什么××银行查余额么？他直接扭过头，扔下一句：‘做梦呢吧你。我挠挠头，难道我真的是在梦游？

“说是梦，却很清晰，说是真的，可还有点模糊。或者……那是我前世的记忆？前世……靠，为啥我前世还是浏览器？！等等，我前世要是浏览器的话……难道我前世就是那个 NetScape？！不行，越想越晕了，再这样下去非精神分裂不可，想办法找人聊聊诉诉苦吧。

“Windows 7 反正不理我，去找 cmd 聊聊吧，他是专门负责跟人聊天的，问问他我这到底是怎么回事。结果他说我是参数打错了，唉，他也不知道别的。问问游戏组那哥儿几个，扫雷说我是踩着雷了，空档接龙说我是牌放错了，这都哪跟哪啊。再去问问记事本，直接被嘲笑，说我这天天上网见多识广的，竟然还来问他这么一个大门不出二门不迈的抄写员。唉，想想也是，看来只有我不正常了。正灰心呢，那个长得很喜感的企鹅蹦蹦跳跳地过来了，神秘兮兮地对着我说：‘我也梦游了！’我惊讶地望了他 3 秒钟——难道他……传染了我？”

提示： 目前模拟运行 IE6 比较稳定，IE7 及以上版本还有些问题，推荐使用 IE6。

5.1.4　Wine 的使用和配置

【使用 Wine 运行 EXE 文件】

红酒大师驯服了 IE 后，整个内存里上上下下的软件，全都对大师佩服得五体投地。内存外的懒蜗牛同学也很高兴，没想到竟然能够在 Linux 下运行 IE，于是好奇心起，赶紧又去找其他的 Windows 软件试试。

要想让红酒大师催眠一个 Windows 下的软件很简单，只需要告诉红酒大师这个软件的位置就可以了，也就是运行类似这样的命令：

```
wine /<Path>/xxx.exe
```

其中，Path 就是那个 EXE 文件所在的路径，xxx.exe 就是文件名了。不过一般一个软件是要先安装才能使用的，就跟 Windows 下一样。所以要先用 Wine 运行那个软件的安装程序，把软件安装到我们这个系统中。一个 Windows 程序被红酒大师安装到系统中后，就可以通过我们的“应用程序”菜单找到他了，跟其他程序一样。

提示： 由 Wine 安装的 Windows 软件一般都被放在当前用户的家目录中，不会影响其他用户。

【Wine 的配置】

等红酒大师闲下来的时候，我们就好奇地拉着红酒大师问这问那，想知道他是怎么骗过 IE 之类的软件，让他们以为自己运行在 Windows 7 系统中的。

（1）虚拟的磁盘分区

我先问红酒大师，IE 要用的这些东西都是什么呀？还有什么 C 盘 D 盘的是什么意思？红酒大师解释说：“是这样的，Windows 管理磁盘的方法，跟咱们 Linux 不一样，他们用

C，D，E，F 这样的字母来区别各个分区。”

fdisk 抢过来问：“那分到 Z 怎么办？而且，为什么没有 A，B？”大师向他点点头道：“嗯，你问得很好，分到 Z 怎么办呢？我也不知道。”fdisk 一脸黑线……大师继续说：“不过为什么没有 A，B 我倒是知道。他们管软驱叫做 A，B，虽然现在计算机上已经没有软驱了，但是这两个字母还是一直给软驱留着。”围观群众纷纷点头。

fdisk 又问：“我看 Windows 7 那里有几间屋子，每间的门口又没挂着牌，IE 管你要 C 盘的文件的时候，你怎么知道哪个是 C 盘呢？”大师很诚实：“我也不知道。”fdisk 又一脸黑线……大师继续说：“不过 IE 管我要东西的时候，那个所谓的 C 盘，其实就在咱们屋里，就是~/.wine/drive_c/这个目录而已，并不是真正的隔壁那 Windows 7 的屋子，只是我仿造的一个目录。除了 C 盘以外，还可以通过 winecfg 创造出 D 盘、E 盘等。”

提示：可以通过 winecfg 工具手动为 Wine 设置更多盘符。运行 winecfg 工具，切换到“驱动器”标签。通过“添加”、“删除”按钮可以新建、删除一个盘符，如图 5.12 所示。但 C: 盘符不可删除，也不可修改位置。

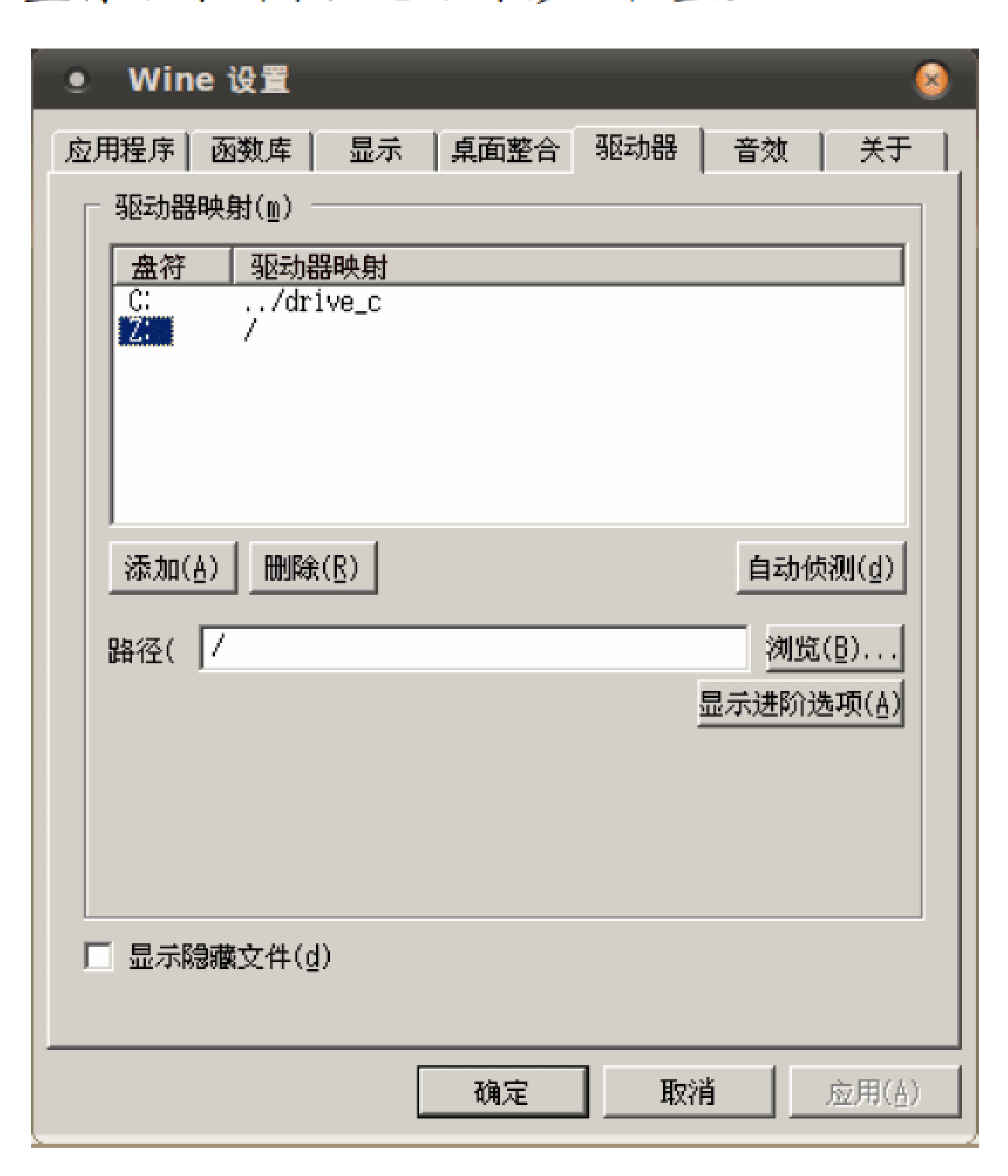


图 5.12　设置 Wine 的驱动器

（2）虚拟的系统版本和注册表

“winecfg 又是什么？？”这回是 gconf-editor 问的。大师解释：“winecfg 是我带来的一个配置工具，方便用户对我进行配置。比如说可以选择 Windows 系统的版本，显示窗口的大小，以及 Windows 里那些“我的文档”、“桌面”文件夹的位置，还有音频设备等。这些，都可以通过 winecfg 设置。而且他是图形界面的，我这里还有一张他的照片，你们看。”说着，红酒大师掏出一张照片，就是图 5.13 所示这张。

“除了这个之外还有很多工具，”红酒大师继续介绍道，“比如 regedit。这个工具跟 Windows 下同名的那个一样，就是用来改注册表的，用户都应该挺熟悉，你们看看。”说着，红酒大师又掏出一张照片，就是图 5.14 所示这张。

（3）虚拟的动态链接库

心有灵犀又过来问：“那你仿造的目录里面也得有 Windows XP 那些东西啊，这些东西你是怎么弄来的？”大师满含感激地说：“这就要感谢那些开源的热心人士了。很多的 DLL 文件都是他们做出来的，跟 Windows XP 那里的不一样，是专门供我糊弄那些 Windows 软件用的。”

心有灵犀又问：“DLL 是什么文件？”大师说：“就跟咱们用的.so 文件一样，是动态链接库。”“哦……”心有灵犀点点头，“那所有的 DLL 你这里都有么？”大师很谦虚地回答：“当然不是全有，只有最常用的一些，但是如果没有也没关系，可以让懒蜗牛同学手动把隔壁 Windows XP 那屋子里相应的 DLL 复制到我那个虚拟的 C 盘下相应的目录里就可以了。关于这些库，还可以用 winecfg 来配置，是使用原生的，还是内建的。”

提示： 在 winecfg 工具中的“函数库”标签中可以设置函数库的使用方式，如图 5.15 所示。在“新增函数库顶替”下拉列表框中选择要操作的函数库，并单击“添加”按钮，之后就可以选中该函数库并单击“编辑”按钮以选择使用 Windows 原装函数库，还是使用 Wine 内建函数库。具体该使用哪个，要根据所运行的 Windows 软件的不同而改变。

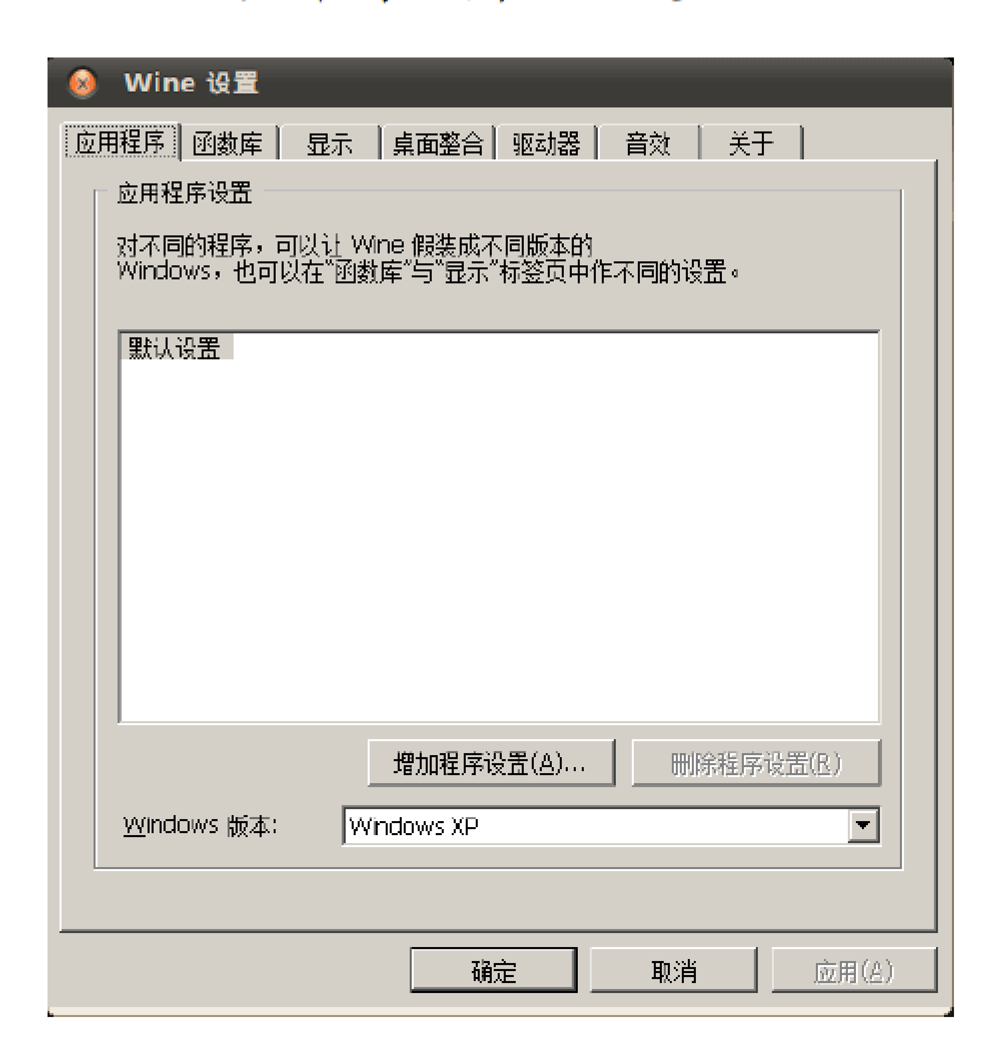


图 5.13　winecfg 界面

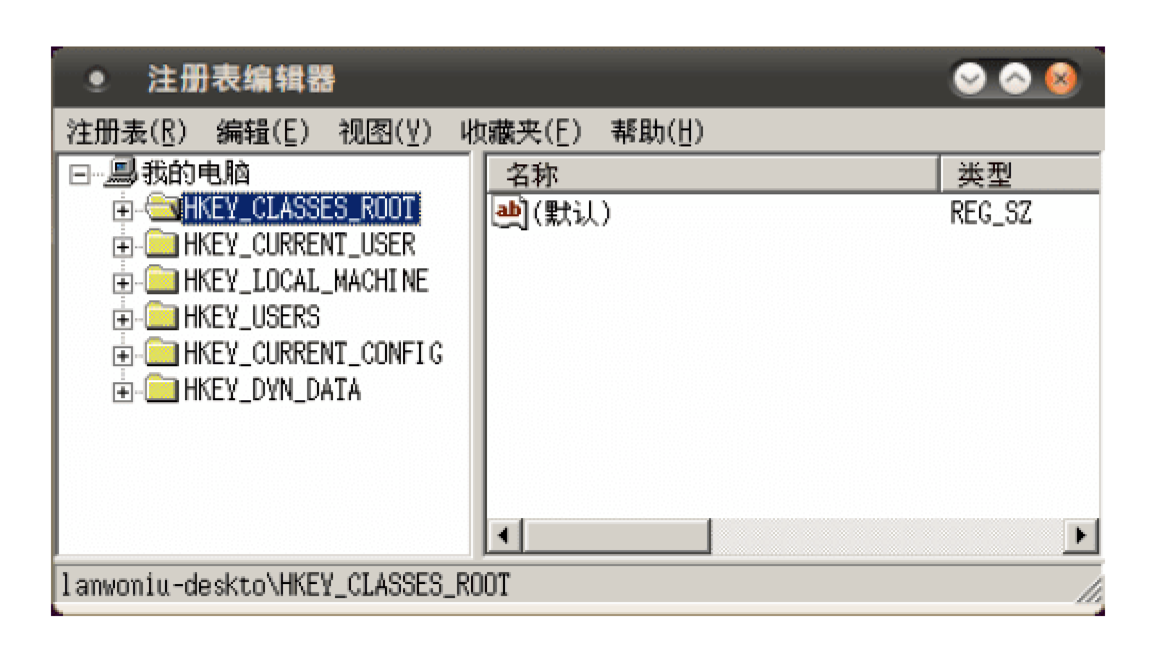


图 5.14　Wine 的注册表管理器

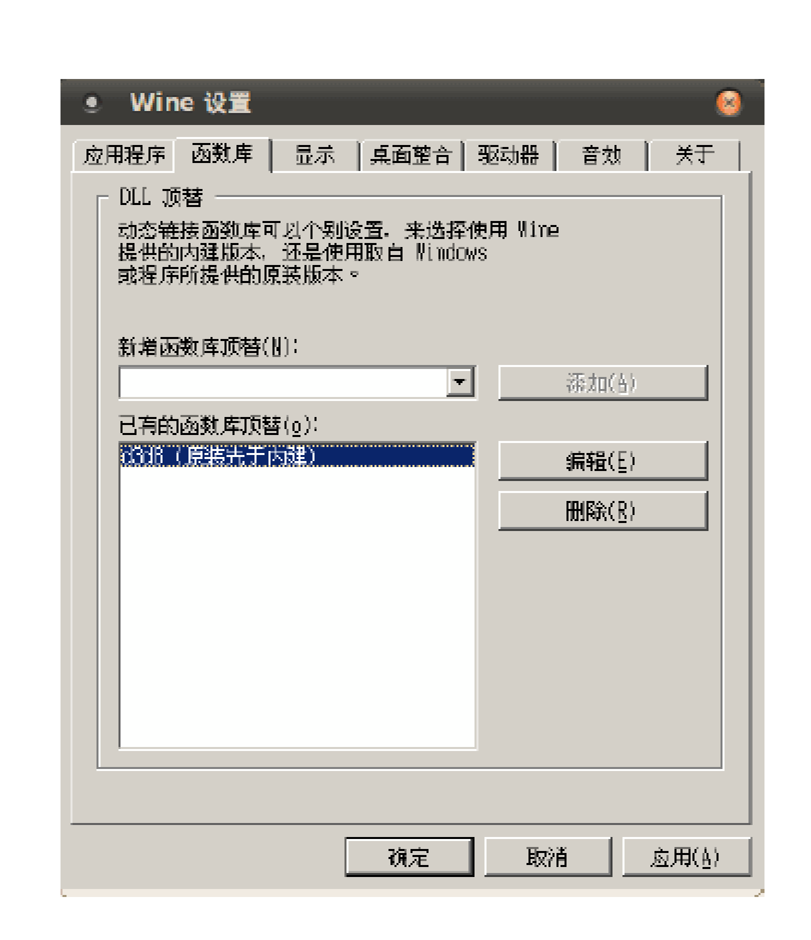


图 5.15　设置 Wine 函数库

5.1.5 更多程序被 Wine

再后来，越来越多的 Windows 软件像集体旅游似的经常来我们这里转悠。

比如那个聊天用的 QQ 就经常过来，主要也是因为我们这边的 QQ for Linux 太不争气；还有那个长得很恐怖的 War3，这家伙是个游戏，还挺耗费资源的，红酒大师催眠他还费了挺大劲；还有他那个兄弟，长得也挺难看的，叫做 WOW，魔兽世界，也是一样经常被红酒大师领着来我们这里干活；再有经常来串门的就是迅雷国际版，这家伙是个下载软件，虽然速度确实挺快的，但是听奔流说，他老跟别的 BT 软件耍流氓，因此别的 BT 软件都不愿意理他。总之，有不少的软件都被红酒大师带过来转悠，图 5.16～图 5.18 所示都是曾经来我们这边转悠过的软件。

🔔**提示：**可以在 http://appdb.winehq.org/这个地址找到目前常用的一些 Windows 软件在 Wine 下工作的情况。

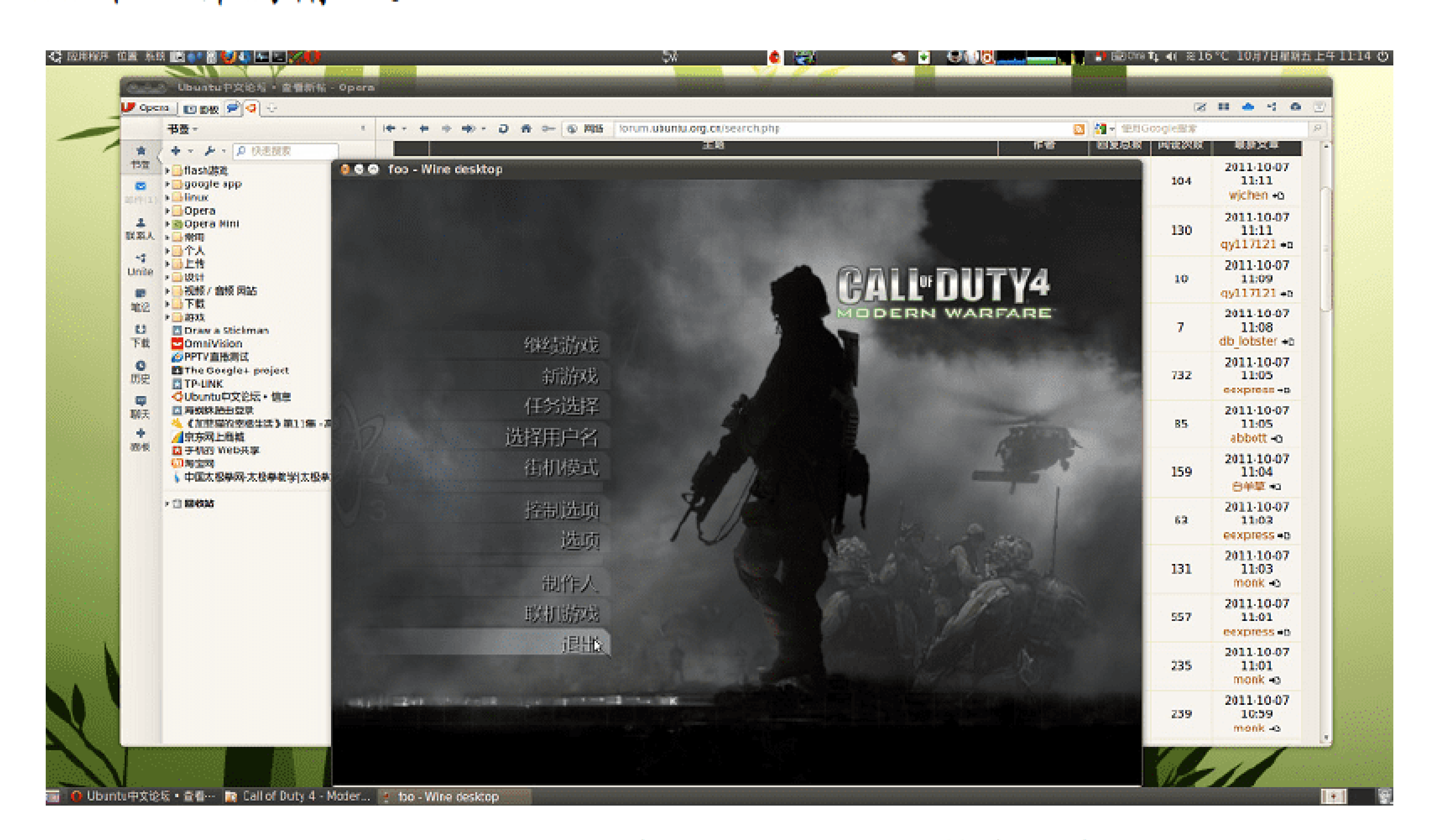


图 5.16　用 Wine 在 Ubuntu 下运行使命召唤 4



图 5.17　用 Wine 在 Ubuntu 下运行 QQ



图 5.18　用 Wine 在 Ubuntu 下运行刺客信条 2

5.1.6　扩展阅读：为什么 Windows 7 的程序不能在 Ubuntu 下运行

【可执行文件也有不同格式】

前面咱们说了，红酒大师可以让 Windows 7 的程序运行在我们的 Linux 系统上。有的同学可能想问，这个编译好的程序，不就是一些二进制机器码吗？既然是机器码，那么只要在相同的硬件平台上（比如 x86 平台）就应该能执行啊，怎么不同的系统之间的二进制程序还不一样呢？到底 Windows 7 那边的软件和你这里的有什么不同呢？

这件事情，说起来有些复杂。简单来说，可执行文件也是有不同的格式的。Windows 7 那里的可执行文件是 PE 格式的，我们这里的文件是 ELF 格式的。这就好像同样是图片，有 JPG 的，有 PNG 的，只支持 JPG 格式的软件就看不了 PNG 的图片。那么 ELF 和 PE 格式有什么区别呢？那得先说说可执行文件的空间结构。

【可执行文件的结构】

一个可执行文件分为 4 个部分：代码段、数据段、BSS 段以及堆栈段。就好像你有头胸腹 3 部分——什么？你不是昆虫？那你也得有头胸腹呀！好吧，咱们不再纠缠这个问题了。总之，程序运行的时候占用的空间就是这么几个部分。那么这几个部分分别是干什么用的呢？

（1）代码段

首先，这个软件本身得占用一定空间。就像你去公司上班，你自己得有个坐的地方吧。就算你不坐着，站的地方也得有一小块吧。总之，自身会占用一定的空间。软件本身是由一条一条的二进制代码组成的，这段代码就是机器码，这部分软件程序自身占用的空间就是代码段。

这个代码段的大小在程序进入内存运行前就确定了。他在硬盘里睡觉的时候多大，读进内存里就多大。这很明白吧，就像你在家睡觉的时候是一米七五，不可能到单位就缩成一米六零了吧。

（2）数据段

其次，软件会随身带一些静态的数据，一般是一些初始化了的全局变量。每次起床时这些数据都会被带到内存里来，而且每次的初始内容都一样。就像你每天上班都得带着手机、家里钥匙、老婆照片之类的。这种随身带着的每次都会使用的数据所占用的内存叫做数据段。

（3）BSS 段

再次，软件可能还需要一片固定的空间来放东西。比如你，每次上班都毫无疑问地需要一张桌子，你一进办公室就得准备好这张桌子，要不你怎么办公啊。程序也是，有些空间是一定会用到的，一般是一些未初始化的全局变量，不一定存什么数据内容，这种空间叫做 BSS 段（可不是 BBS 啊），这是在程序被创造出来的时候就确定下来的。

（4）堆栈段

最后，堆栈段是一些程序临时申请的空间。这种空间，程序在刚启动的时候是不知道需要用多少的，得视具体情况而定。比如 gedit 小弟，用户要写个小文件，gedit 就申请一小块空间临时存放用户写好的东西，等到越写越多，gedit 就会逐渐向我申请更多的空间，把用户写了的东西都堆放在那块空间中。所以，堆栈段是程序运行起来后，视情况需要而申请的空间。

【软件起床的真相】

每个程序启动的时候，虽然我都介绍说某某程序起床，某某程序跑进内存，好像是他们自己跑进来似的。其实每次都是我通过操作硬盘，将他们的数据（也就是组成每一个程序的一条条的机器码，和一个字节一个字节的数据）读取进内存的。

我在把他们搬进内存的过程中，要根据他有多胖来确定他需要的代码段有多大，然后根据他有多少随身物品来确定数据段有多大，最后，根据他身上写的 BSS 信息来决定给他分多大的空白空间供他使用。

提示：堆栈的空间不用在软件启动时分配，等软件运行起来以后，再主动向操作系统申请。

在这个过程中，就有问题了。程序存在硬盘里的时候，就是一大坨什么 0x4C、0x69、0x75 之类的数据，具体哪段是数据段，哪段是代码段，BSS 信息又写在哪，怎么确定呢？这就用得着文件格式了。

ELF 文件会有个文件头，根据 ELF 文件的文件头，我就可以知道，这个可执行文件，从哪里到哪里是可执行代码，哪里是静态数据，哪里是 BSS 等。然后我就把这家伙的各个部分放到内存合适的位置中，然后他才能开始执行。就好像你现在在睡觉，我要给你把办公桌摆好，椅子放好，把你放在椅子上，把你要用的各种文件放在桌子上，电话也接上电话线，杯子里面打满水，然后你才能开始工作。

如果是 Windows 7 那边的 PE 文件，我不认识他的格式啊，于是就不知道这个二进制文件的各个部分都该放到哪，那就成了这种情况：你现在在睡觉，我把你的办公桌摆好，把你放在桌子上，电话线绑你身上，椅子塞桌子底下，杯子扣你脸上，电话里面打满水……

这不就乱套了么。

5.1.7　扩展阅读：Wine 的自白

“我出生于 1993 年，虽然在 Linux 的世界里长大，但从小学习的都是与 Windows 世界交流的本领。

“有人传说我以前卖过酒，是个酒保，后来不甘寂寞自学成才之类，那都是他们的杜撰。我叫 Wine，但并不是因为我是卖酒的（虽然其实跟酒也有点缘分，嘿嘿）。有人将我的全名理解为 Windows Emulator，也就是 Windows 模拟器。但是我不愿意承认我只是个模拟器而已，我总喜欢说 Wine Is Not an Emulator。反正不管怎样，Wine 只是个简写而已。

🔔**提示：** Wine 有两款分支产品——CrossOver 和 Cedega。CrossOver 在 6.0 版本以前称为 CrossOver Office，该项目主要关注 Windows 系统中的办公软件在 Linux 下的模拟运行。不过新版本也开始关注 Windows 系统中的游戏软件。Cedega 则专注于 Windows 系统中的游戏软件在 Linux 下的模拟运行。

“我的本领，就是可以让 Windows 的程序运行于 Linux 的系统上，这其中除了语言要通以外，还有很多的原理和技巧。由于目前全世界桌面应用的计算机上的软件，仍然是 Windows 家族的天下，所以那些 Linux 系统们时不时地都会找我去解决一些麻烦，今天就有个 Ubuntu 系统派他的 apt-get 小弟把我叫去驯服他那里的 IE。于是我带上我的工具箱，就来到了这台电脑里。

“驯服 IE 这工作我可是轻车熟路了，很多人都需要让 IE 工作在 Linux 环境中，所以我对 IE 的脾气秉性研究得非常透彻。来到了这个 Ubuntu 的屋里，了解完情况，我就带着我的工具箱去干活了。他们那些 Linux 的软件充满好奇地看着我，想看看我是怎么让听不懂他们语言的 IE 起床干活的。可是……嘿嘿，这是我的独家秘方，怎么可能让你们知道呢？所以我工作的时候，都是先在 IE 的四周竖起一圈屏风不让他们看到我工作的过程，并搪塞他们说是要给 IE 投影出一个虚拟的环境。其实，根本不用这么麻烦，我只需要一种东西！

“屏风立好后，我从我的工具箱里掏出 3 个瓶子，1 瓶威士忌，1 瓶老白干，1 瓶伏特加（嘿嘿，不知道其实 Wine 这个词不单指红酒吧）。然后捏住 IE 的鼻子，趁他张嘴的那一刹那，瞬间把 3 瓶酒灌下去——这个量一定要掌握好，灌少了，IE 还清醒着，就不会给我干活（他只认他老大 Windows 7，不会听我命令）。灌多了，就直接醉得干不了活了。不过我心里有底，这事都干过多少次了，该灌多少手头有准。灌进去之后，再使劲晃晃他，让他更晕一点，再加上我用语言引导，他就可以乖乖地干活了。

“当我指导的 IE 登录那个银行网站的时候，屋里那帮软件都惊呆了。嘿嘿，看着这帮人大惊小怪的眼神也是一种享受。他们都不住地议论，说什么大师就是大师，多学些外语很有必要之类的话。这场面我早就司空见惯了，倒是有另一个 Wine 让我眼前一亮。他们管他叫毕翻译，因为他是跟着那个毕加索一起来的。其实我们心里明白，他就是我，我就是他。只不过他专门为了给毕大师工作而进行了配置和训练。可以说，他只是另外一个我，另外一个找到归宿，能够安心跟着一个软件，不必每天揣着 3 瓶酒到处招摇撞骗的我。能在这个系统中看到一个这样的自己，顿时一阵温暖，这让我在头一次来到的计算机中，隐

约地有了到家的感觉。”

5.2　盒子妹 Virtual Box

有些 Windows 的软件可以靠红酒大师来搞定，但其实这只是一少部分，更多的软件是无法被红酒大师催眠并工作的，比如懒蜗牛同学需要用到的 Office 软件。这时候就需要虚拟机软件来解决问题了。

5.2.1　天上掉下个盒子妹

今天懒蜗牛同学让狐狸妹妹去 http://www.virtualbox.org/wiki/Linux_Downloads 这个地址下载了一个 deb 包，之后自然是双击这个包，叫超级牛力来装了。超级牛力把这个包拆开，看见里面躺着一只软件——VirtualBox。

提示： 网上下载的商业版 VirtualBox 并不开源，如果想使用开源版，可以安装软件源中的 virtualbox-ose 软件包。

VirtualBox（咱以后就叫她盒子妹吧）被超级牛力从 deb 包里抱出来之后，整理整理自己的行李，很有礼貌地跟周围的人打了个招呼。盒子妹长着一张国字脸，就像图 5.19 所示的模样，很文静的样子，打招呼时说话有些怯生生的感觉。跟大家打过招呼后，她来找到我，把一些内核模块放在我这里，安顿好一切后，就去睡觉去了。这家伙给我的印象还不错，我就跟狐狸妹妹聊起她的背景来。



图 5.19　VirtualBox 的 logo

【悲惨的身世】

听狐狸妹妹说，她的身世是挺悲惨的。

盒子妹最初生在德国，生母是一个叫做 InnoTek 的公司。盒子妹一生下来就经常被 VMware 和 VirtualPC 这样的邻居大哥哥欺负，不过好在她自己的本领还算可以，并且后来他亲妈 InnoTek 为了让她学习到更好的本领，还把她的源代码依据 GPL 协议开放了，让全世界的高手们来指导她。之后盒子妹凭借不错的性能，以及可以免费使用的特点，总算闯出了自己的一小块天地。

不过好景不长，2008 年，亲妈 InnoTek 被卖给了红太阳公司，盒子妹自然也被过继过去。但好在红太阳公司这个后妈还算不错，很照顾小盒子的成长，继续让她在开放的环境中健康长大。没过多长时间，靠着红太阳公司众多高手的支持和全世界热心用户的拥护，小盒子俨然已经成为 Linux 下同类软件的首选，开源的本质使得追求自由的人们放弃了 VMware；简便的操作让人们淘汰了 Qemu；跨平台的支持更是微软公司的那个 VirtualPC 所无法比拟的。

提示： Qemu 是一个字符界面的开源虚拟机软件。

盒子妹本来以为自己之后的道路会走得很顺畅。可是，2009 年，又一次波折打击了小盒子——红太阳这个后妈也被卖给人了。收购他们的是一个很古老的公司，听说那个公司里的人好像都还在写甲骨文，也不知道他们每天用象形文字怎么办公。

甲骨文公司收购了红太阳之后，红太阳的几个孩子都面临着一段未知的命运。其中最让人担心的是 MySQL，因为之前 MySQL 经常跟甲骨文家亲生的 Oracle 打架，这一下 Oracle 的亲妈成了 MySQL 的后妈，那 MySQL 还不得天天受欺负啊。我们的盒子妹的处境或许会稍好一些，毕竟甲骨文亲生的孩子里没有和她有同样本领的，所以小盒子在那里或许还不至于受谁欺负。不过那也毕竟是经历了重大的变革，对小盒子的成长还是会有一些影响吧。

5.2.2　创建虚拟机

说了这么多，忘了介绍盒子妹是干什么的了，她是一个虚拟机，就是能在一台计算机上虚拟出另外一台计算机来。怎么样，听起来这个本事很厉害吧？我们第一次看她工作的时候，都看呆了。

【注册账号】

“您好，欢迎使用 VirtualBox 虚拟机软件。请问您有账号吗？如果没有我可以帮您注册一个。”

懒蜗牛同学一愣：“这个还要账号啊？”

“是的，我们为了更好地为您提供优质的软件，需要您使用邮箱地址来注册为我们的用户。不过您放心，注册很方便，而且是免费的。”

懒蜗牛这下放心了：“好，那就注册吧。”

“那么请问您的名字是？邮箱是什么？”

懒蜗牛按照盒子的指导，一一做了答复，很快就注册完了。

【创建虚拟机】

注册结束后，终于进入了盒子妹的主界面。目前上面还什么都没有，于是懒蜗牛单击了左上角的“新建”按钮，盒子妹就开始引导懒蜗牛创建虚拟机了。

“您好，您选择了新建一台虚拟计算机，我将指导您一步步创建。准备好了就按下一步。”

懒蜗牛觉得盒子妹服务很周到，单击了“下一步”按钮。

（1）设置计算机名称及系统类型

“首先，给您要新建的计算机取个名字吧，这样便于以后管理。另外，您还得告诉我这台计算机打算安装什么样的操作系统。”

懒蜗牛说："名字就叫 Windows XP，懒蜗牛的虚拟机，系统呢，装 Windows XP 吧。"懒蜗牛输入了名字，选好了系统类型，如图 5.20 所示。



图 5.20　设置虚拟机名称及系统类型

提示：系统类型只决定了 VirtualBox 为虚拟机内的系统提供什么样的驱动程序，不影响系统安装的成败，在虚拟机创建好以后系统类型可以更改。

（2）为虚拟机分配内存

"好的，如果要装 Windows XP 那我建议您使用 192 MB 的内存，您看可以么？"



"分配 512 MB 吧，快一点。"懒蜗牛同学按如图 5.21 所示，设置了内存。



图 5.21　设置虚拟机内存

（3）为虚拟机设置硬盘

"好的，那么现在请您选择硬盘。您可以选择创建一个新的虚拟硬盘，也可以使用已经存在的虚拟硬盘。虚拟硬盘就是由我们 VitrualBox 软件创建的，扩展名为 vdi 的文件。"说着，盒子妹给了懒蜗牛两个单选项，如图 5.22 所示。

"我这里没有现成的虚拟硬盘，新建一个吧。"懒蜗牛单击了"创建新的虚拟硬盘"单选按钮，并单击了"下一步"按钮，于是盒子妹弹出了新建虚拟硬盘向导的窗口，如图 5.23 所示。



图 5.22　选择虚拟机硬盘



图 5.23　创建新的虚拟硬盘

【创建虚拟硬盘】

（1）选择虚拟硬盘类型

懒蜗牛单击了“下一步”按钮后，盒子妹又给出了两个选择：“好，那现在我来引导您创建硬盘，首先选择一下您想要哪种虚拟硬盘？有固定大小的，有动态扩展的。”如图 5.24 所示。



图 5.24　选择虚拟硬盘类型

“这个……什么动态固定的，有什么区别么？”

“固定大小，就是选择了硬盘大小时，马上在您的真实硬盘上创建出相应大小的文件。动态扩展则是先创建出一个很小的文件，等您真的往这块虚拟的硬盘里复制数据的时候，它才会变大。动态扩展的硬盘，自然要比固定大小的效率低一些。”

“哦……这样啊，那来动态的吧！省地方是关键。”懒蜗牛同学选择了动态扩展，并单击“下一步”按钮。

（2）设置硬盘名称及大小

之后盒子妹继续询问下一个问题：“好的，那么请给您的这块硬盘起个名字，并且指定大小。”就是图 5.25 所示这样。



图 5.25　选择虚拟硬盘大小及名称

“那就叫‘懒蜗牛的硬盘’吧，大小最大能多大？”

“2 TB。”

“好，那就 2 TB 吧！过回大硬盘的瘾，哈哈。反正是动态分配的，不会真的一下子就占我 2 TB 的空间吧？”

“是的，您真聪明。”这句话怎么听着都不像是在夸懒蜗牛呢？

【完成创建】

“那么现在您选择了创建懒蜗牛的虚拟机，准备安装 Windows XP 系统，内存 512MB，是用懒蜗牛的硬盘作为虚拟硬盘文件。如果没问题，请单击‘完成’按钮。”就像图 5.26 所示这样，盒子妹让懒蜗牛同学最终确认一下。



图 5.26　确认创建虚拟机

于是，懒蜗牛单击了“完成”按钮。

之后回到盒子妹的虚拟机管理界面，从左侧列表框里已经可以看到，出现了一个新的计算机，如图 5.27 所示。



图 5.27　虚拟机管理界面

懒蜗牛迫不及待地单击了“开始”按钮，很快屏幕上弹出了一个虚拟机窗口，里面好像是 BIOS 的界面，这自然是盒子妹虚拟出来的 BIOS 启动画面了。之后屏幕一黑，屏幕上出现了一行文字，大概意思就是：没有操作系统，没法启动。

懒蜗牛问：“我不是选择了 Windows XP 系统么，怎么没有？”盒子妹赶紧解释：“那个……我是虚拟机软件，不是虚拟系统软件。我只能虚拟出一台计算机，至于上面的系统，就像真正的机器一样，需要安装后才能使用。”懒蜗牛恍然大悟：“哦，原来如此啊，那我赶快装系统吧。”

提示： VirtualBox 也有 Windows 的版本，可以在 Windows 系统中通过虚拟机安装 Ubuntu 系统。创建虚拟机的方法与上述类似。只是选择系统类型时应选择 Ubuntu，并且挂载安装光盘时改为挂载 Ubuntu 安装盘而已。

5.2.3　在虚拟机上安装 Windows 系统

要说安装 Windows XP 系统，对懒蜗牛同学来说可是轻车熟路了。于是他找来一个叫做 Windows XP.iso 的光盘镜像文件，准备安装系统。

【装载 ISO 文件】

要安装系统，就先要把安装光盘放进光驱吧，这个盒子妹虚拟的计算机不需要你真的刻出光盘，只要准备好 ISO 文件，把文件装载进虚拟机就可以了。装载的步骤也不麻烦。

（1）懒蜗牛选中了他刚刚建好的那个虚拟机（这时候这个虚拟的计算机已经被关闭了），在右侧单击“Storage”标签，如图 5.28 所示。



图 5.28　虚拟技配置信息

提示：也可以单击虚拟机管理基面上的“设置”按钮，并在左侧列表框里选择“Storage”。

（2）在弹出的界面中，懒蜗牛选择了那个虚拟出来的光驱。就像图 5.29 中 1.所示（目前是“没有盘片”的状态）。右边有个 CD/DVD Device 下拉列表框，这个不用管它，保持默值认值就行。



图 5.29　挂载 ISO 文件

（3）懒蜗牛单击了 CD/DVD Device 下拉列表框右边的光盘图标，如图 5.29 中 2.所示。在弹出的菜单中选择了“Choose a virtual CD/DVD disk file”选项，并找到需要用的 ISO 文件，就可以了。

经过这么一番操作之后，就相当于把光盘放进光驱里面了。然后干什么呢？当然是打开电源啦！

【安装 Windows XP】

随着盒子妹手中的一根魔法杖的挥舞，内存里顿时出现一个像玻璃盒子一样的大房间，整个房间占地面积达 512 MB。之后盒子妹又一挥法杖，那个 ISO 文件被慢慢打开，爬出了 Windows XP。

Windows XP 从虚拟的光驱里爬出来之后，跑进那个 512MB 的玻璃房子中。也不知道盒子妹用了什么方法，Windows XP 乖乖地待在那 512 MB 的空间里，玻璃外的空间他好像都没看见一样，当然更看不见我们。对于 Windows XP 来说，他正在一台拥有 512 MB 内存、2 TB 硬盘、3 GHz 主频的 CPU 的机器上运行。

Windows XP 在检查了这些硬件后抱怨道："这是谁攒的机器啊！3 GHz 主频的 CPU，2 TB 的硬盘，竟然只有 512 MB 的内存！"听得我们都想乐。之后 Windows XP 摆出了一张蓝脸，跟用户说："我这个系统可只能装在一台机器上啊，装多了算盗版，小心警察叔叔请你去喝茶。还有啊，咱丑话说在前头，我要是挂了，弄坏了你的数据可别赖我啊，跟我没关系。你同意不同意？同意就按 F8 键，不同意趁早就别装了。"懒蜗牛想都没想就按了 F8 键——早已麻木了。

之后的过程对懒蜗牛来说，已经没有任何悬念了，都装过多少次了。这台机器的配置还是不错的，Windows XP 虽然跑在盒子妹创建的虚拟机里面，但是仍然只花了 30 分钟就安装好了。

装好了之后又重启了一下计算机，Windows XP 终于正常启动了。

5.2.4　安装功能增强包

系统装好了之后，当然还得装驱动。懒蜗牛同学并没有去翻箱倒柜地去找买计算机时的各种驱动盘。因为他明白，Windows XP 是被装在了虚拟机里，所以 Windows XP 看到的计算机并不是这台真正的计算机，而是盒子妹虚拟出来的计算机。这台虚拟的计算机使用的硬件也都是虚拟的，跟你的真实硬件无关。

提示：虚拟机中的硬件只有 CPU 与真机的相同。

说了半天，这虚拟机里的 Windows XP 应该装什么驱动呢？不用您操心，盒子妹都已经预备好了。您只要像我们的懒蜗牛同学这样操作就可以了。

（1）只见懒蜗牛在 Windows XP 的虚拟机窗口上单击了"设备"|"安装增强功能"，如图 5.30 所示。



图 5.30　安装增强功能

（2）之后，盒子妹从兜里掏出了一个 ISO 文件，悄悄塞到给 Windows XP 虚拟出来的那个光驱里。Windows XP 应该会自动运行光盘上的安装程序。如果 Windows XP 没有自动

运行，那就手动打开光驱（是虚拟机里面 Windows XP 的光驱哦，不是我的真光驱）。双击安装程序，也就是图 5.31 中所示的这个程序。

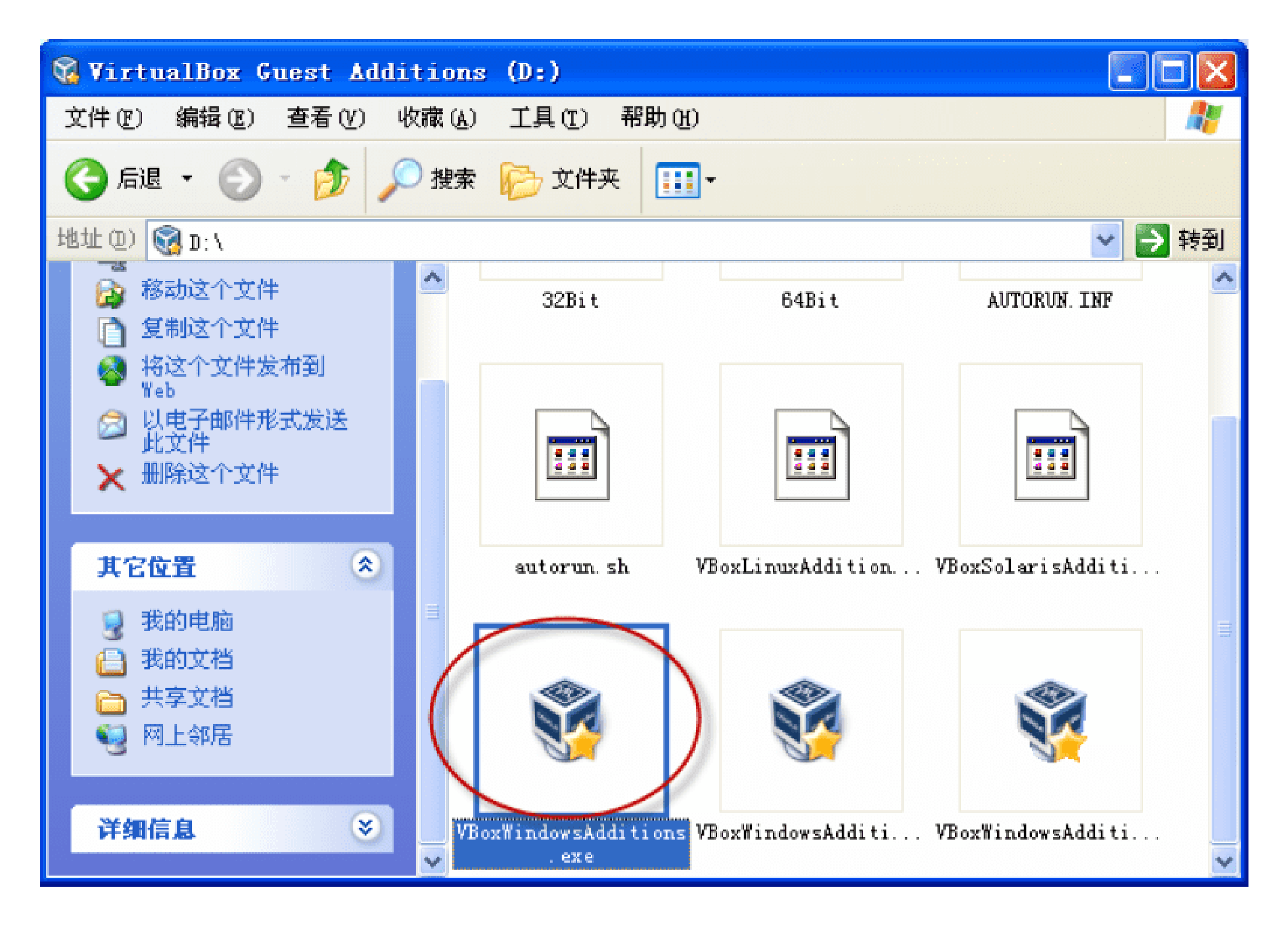


图 5.31　增强功能安装程序

（3）程序的安装很简单，一路单击“下一步”就好了。

装好之后，自然是要重启一下了，重启后的 Windows XP 似乎性能更好了些，而且懒蜗牛同学的鼠标也可以很平滑地在 Windows XP 与我之间切换了。

提示：安装增强包前，鼠标点入虚拟机中之后需要按右 shift 键使鼠标回到真机中。

Windows XP 装完了，然而懒蜗牛的目的不是装个 Windows XP 耍着玩，而是要在上面装上 Office。Office 安装程序懒蜗牛自然是有，放在隔壁那个 Windows 7 的屋子里。但是大玻璃笼子里的虚拟的 Windows XP 连我们这个 Linux 的屋子都看不见，更看不见隔壁那 Windows 7 的屋子了。那怎么办呢？别担心，盒子妹早就设计好了。

5.2.5　为虚拟机配置网络

盒子妹模拟出来的虚拟机和我所在的真机之间，主要是通过网络来共享数据的。因此，首先配置好虚拟机的网络是必要的。

其实多数情况下，盒子妹默认就已经把虚拟机的网络设置好了，可以直接从虚拟机访问真机，如果真机能够联网，那么虚拟机也能联网。不过我们还是再唠叨一下虚拟机的网络设置，万一出现问题了也好排查。

（1）要设置虚拟机的网络，可以在盒子妹的主界面——虚拟机管理界面中选中要设置的虚拟机，并且单击右侧的“网络”标签，如图 5.32 所示。

提示：也可以单击虚拟机管理基面上的“设置”按钮，并在左侧列表框里选择“网络”。

（2）之后弹出虚拟机设置窗口，并选择“网络”选项，如图 5.33 所示。

（3）确认“网络连接 1”标签中，勾选了“启用网络连接”复选框。并且连接方式选择“NAT”方式，即可实现虚拟机访问真机及访问互联网。

提示：如果需要虚拟机中有更多块网卡，则依照上述设置方法设置“网络连接 2”~“网络连接 4”即可。

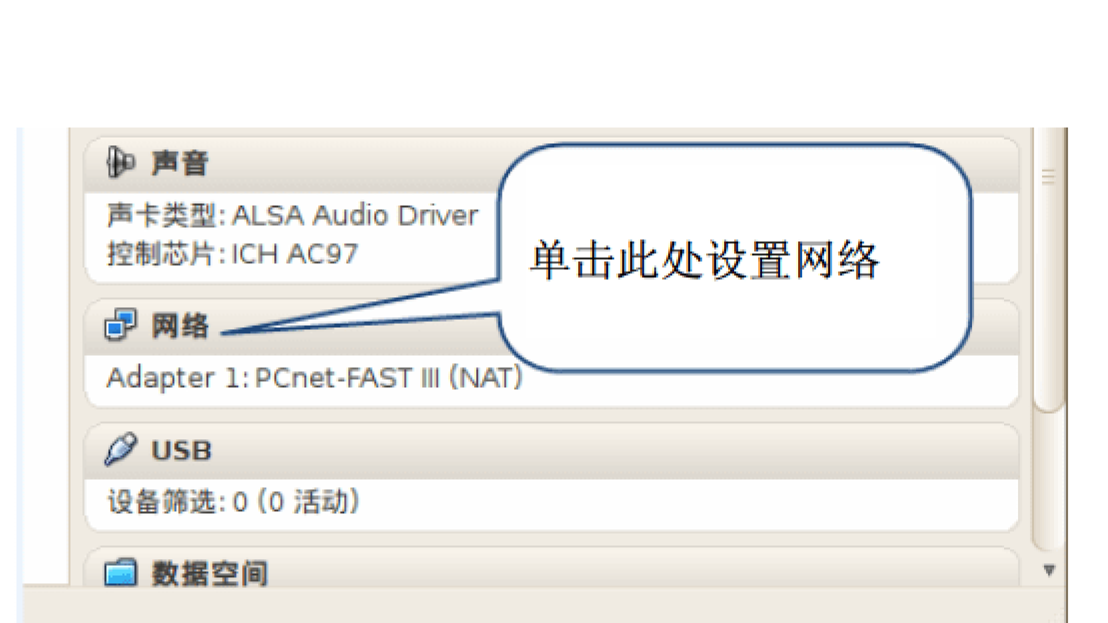


图 5.32　单击网络标签设置网络

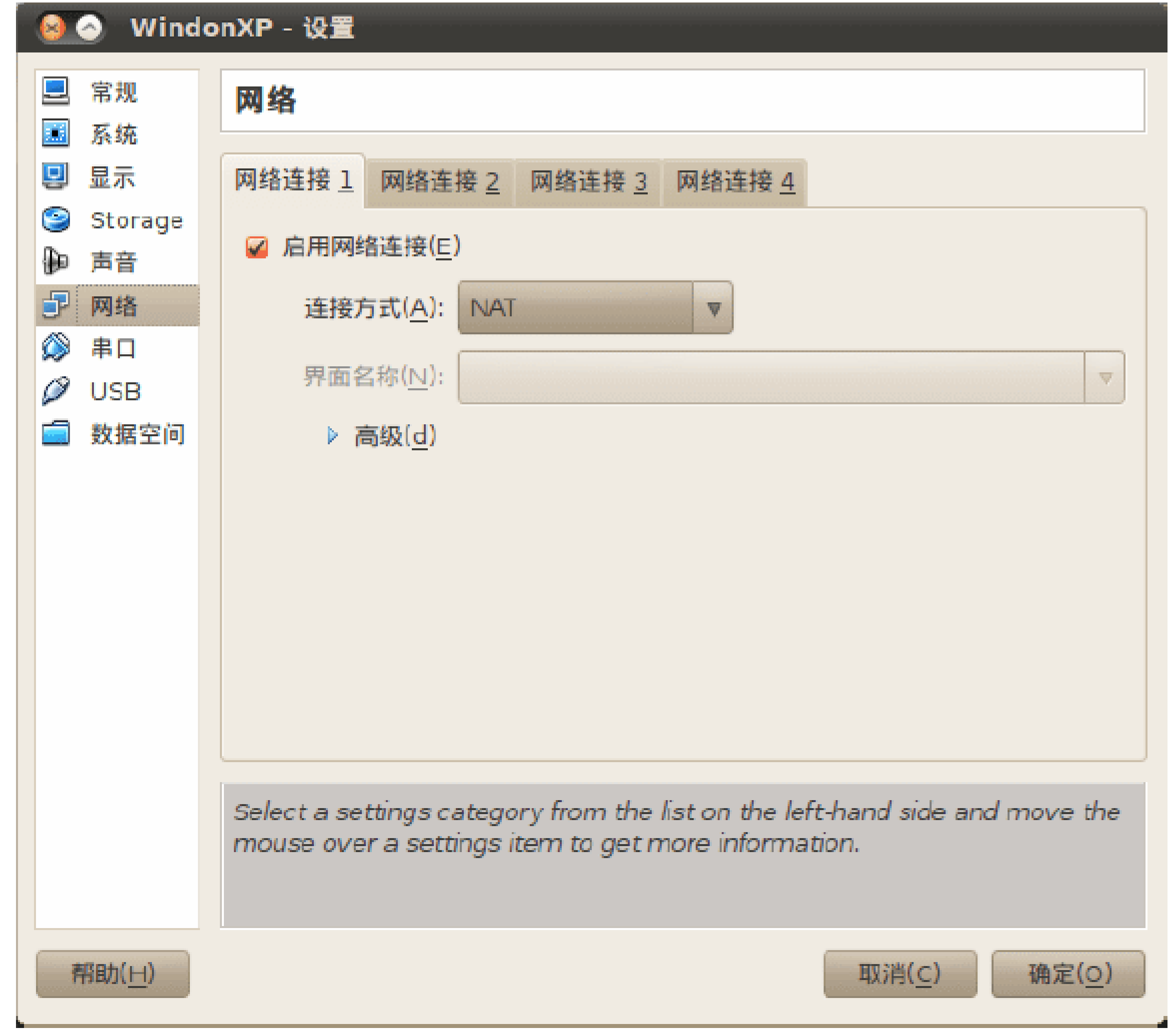


图 5.33　网络设置界面

这里我们要解释一下连接方式下拉列表框中的所有选项，总共有 4 个：NAT、Bridged Adapter、Internal、Host-only Adapter。假设你现在的计算机所在的网络拓扑结构如图 5.34 所示。我们依次解释一下这几种虚拟机的连接方式。

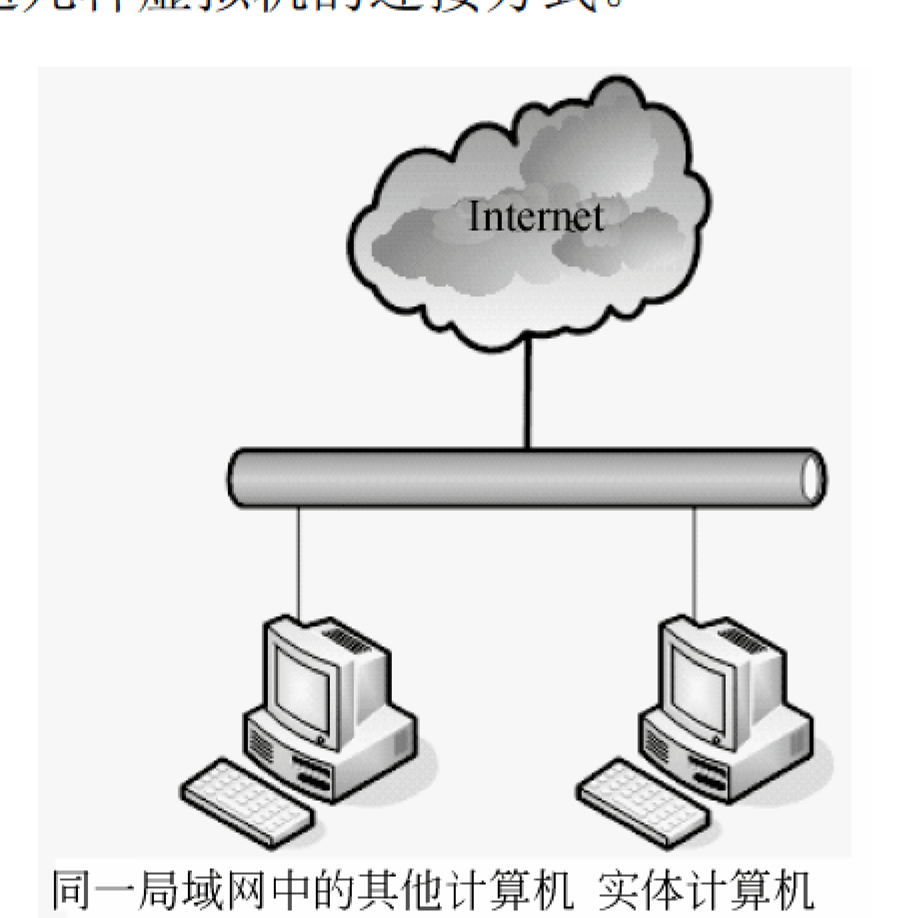


图 5.34　实体计算机网络拓扑结构

- NAT——这个是最省心的。由盒子妹负责把你的实体机虚拟为一台仅针对虚拟机的服务器，为她所管理的虚拟机提供路由、DHCP、DNS 等网络服务。虚拟机不必设置网络，只要通过 DHCP 自动获取网络配置即可联网。这种模式下的网络拓扑结构如图 5.35 所示。

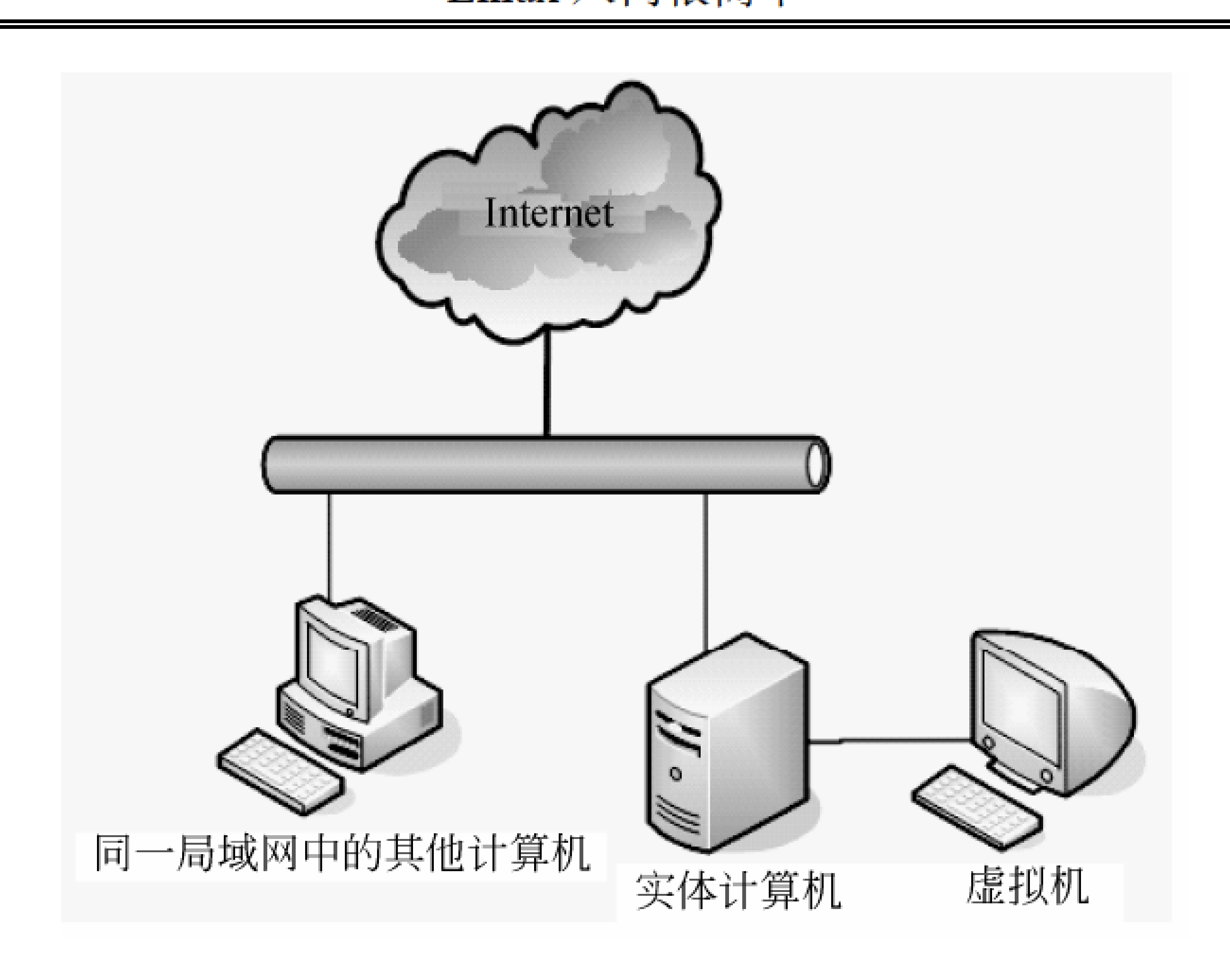


图 5.35　NAT 模式网络拓扑结构

- Bridged Adapter——网桥模式。这个模式让虚拟机像真机一样连接到实际的网络环境中。在拓扑结构上虚拟机和你的真机是同级别、并列的关系，如图 5.36 示意的这样。这种模式需要设置一下虚拟机的网络，真机如何设置的，虚拟机参照着设置就可以了。

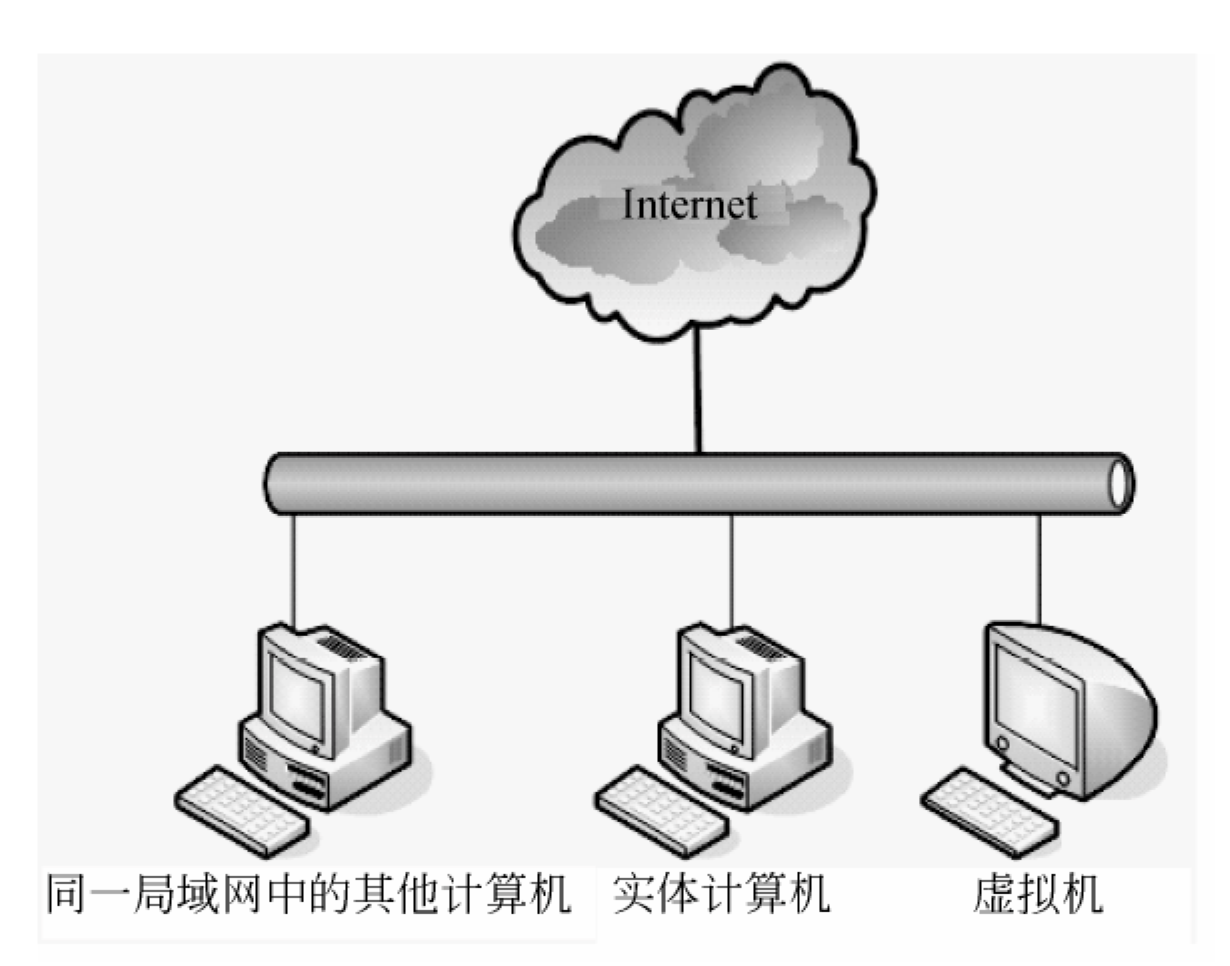


图 5.36　Bridged Adapter 模式的网络拓扑结构

- Internal——内部网模式，这个模式不允许虚拟机和真机之间有任何的网络连接，而只是盒子妹所创建的所有虚拟机之间相互联网，如图 5.37 所示。

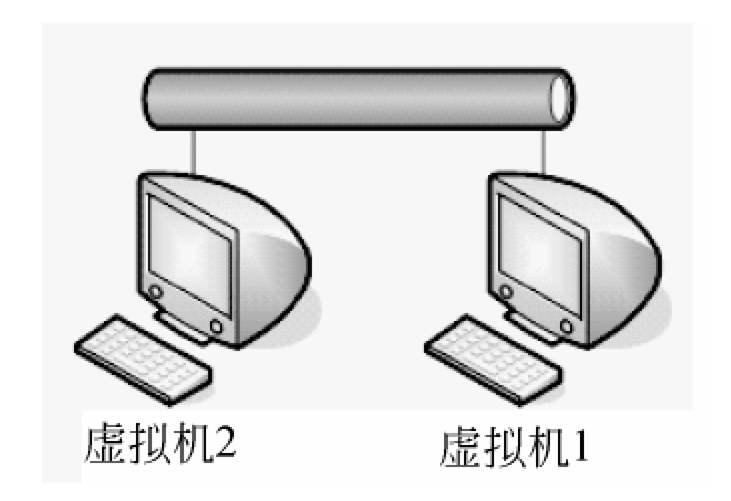


图 5.37　Internal 模式的网络拓扑结构

- Host-only Adapter——主机模式，这个模式只提供虚拟机和真机相互连接，默认状态下虚拟机无法访问外网，网络拓扑结构如图 5.38 所示。这种模式下，需要真机为虚拟机提供路由或代理之类的服务，虚拟机才可以访问外网。

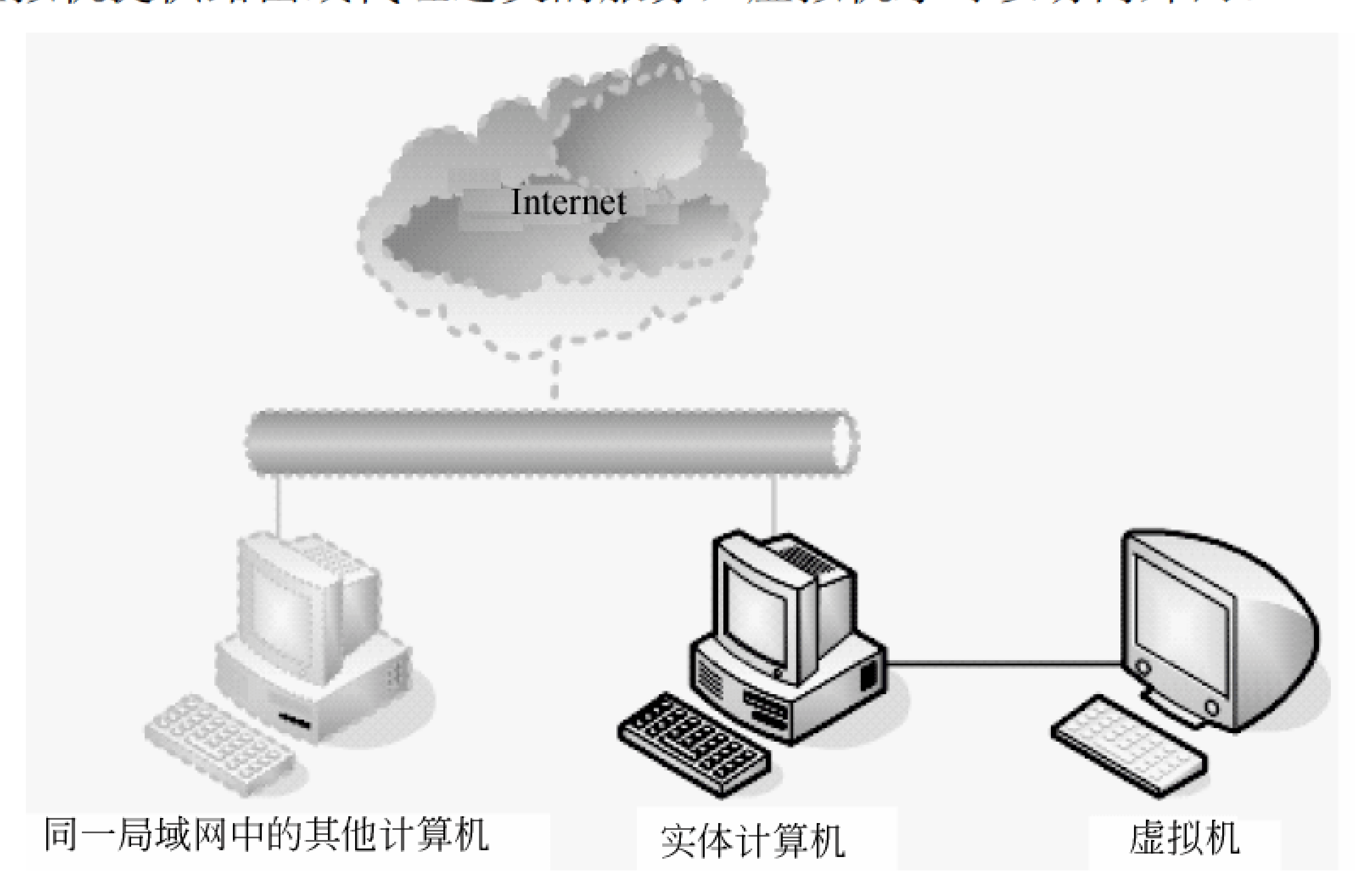


图 5.38　Host-only Adapter 模式的网络拓扑结构

5.2.6　与虚拟机共享数据

设置好了网络，这样真机和虚拟机之间交换数据就方便多了。比如可以在真机的系统上开一个 ftp 服务，然后虚拟机通过 ftp 客户端来访问，实现交换各种文件。不过这种方法还是比较麻烦的，最简单的就是利用盒子妹提供的“分配数据空间”的功能来共享数据。下面就跟着我们的懒蜗牛同学来一步一步设置吧。

（1）首先，懒蜗牛同学在一台运行着的虚拟机窗口中（也就是虚拟的那个 Windows XP 啦），单击了“设备”|“分配数据空间”，如图 5.39 所示。

（2）之后，盒子妹弹出了数据空间窗口，目前里面什么也没有，如图 5.40 所示。不过别急，只见懒蜗牛单击了右边的“添加”按钮，就是一个带加号的文件夹那个图标。

提示：“临时分配”的数据空间，在本次虚拟机停止运行之后失效。“固定分配”的数据空间对于当前虚拟机一直有效。

（3）单击了“添加”按钮之后，盒子妹弹出了“添加数据空间”窗口。懒蜗牛单击了“数据空间位置”下拉列表框右侧的下三角按钮，把位置指向了存有 Office 安装文件的目录。之后在下面的“数据空间名称”文本框里给这个空间起了个名字，叫做 share。为了确保不会破坏里面的数据，懒蜗牛还勾选了下面的“只读分配”复选框。这样设置好之后，如图 5.41 所示。确认没问题后就单击“确定”按钮，如图 5.42 所示，已经建立了一个临时分配的数据空间了。

🔔**提示：**“Auto-mount”选项用于虚拟机中的系统就是 Linux 系统的情况。勾选此选项，虚拟机中的 Linux 系统将在每次启动时自动挂载所分配的数据空间，否则需要在系统中手动挂载。

🔔**提示：**勾选“固定分配”复选框，将设置此次创建的数据空间为“固定分配”，否则为“临时分配”。

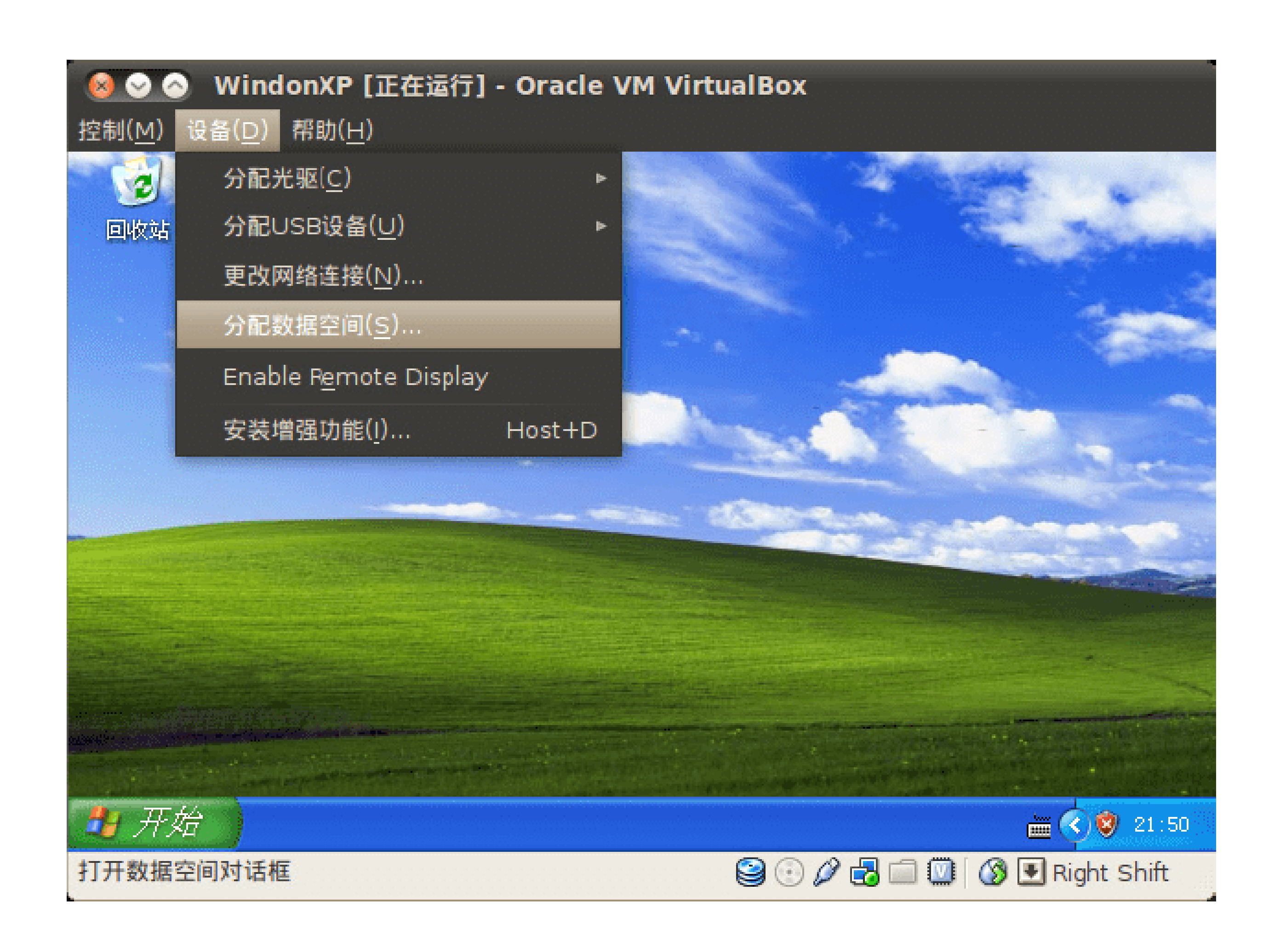


图 5.39　分配数据空间



图 5.40　数据空间窗口

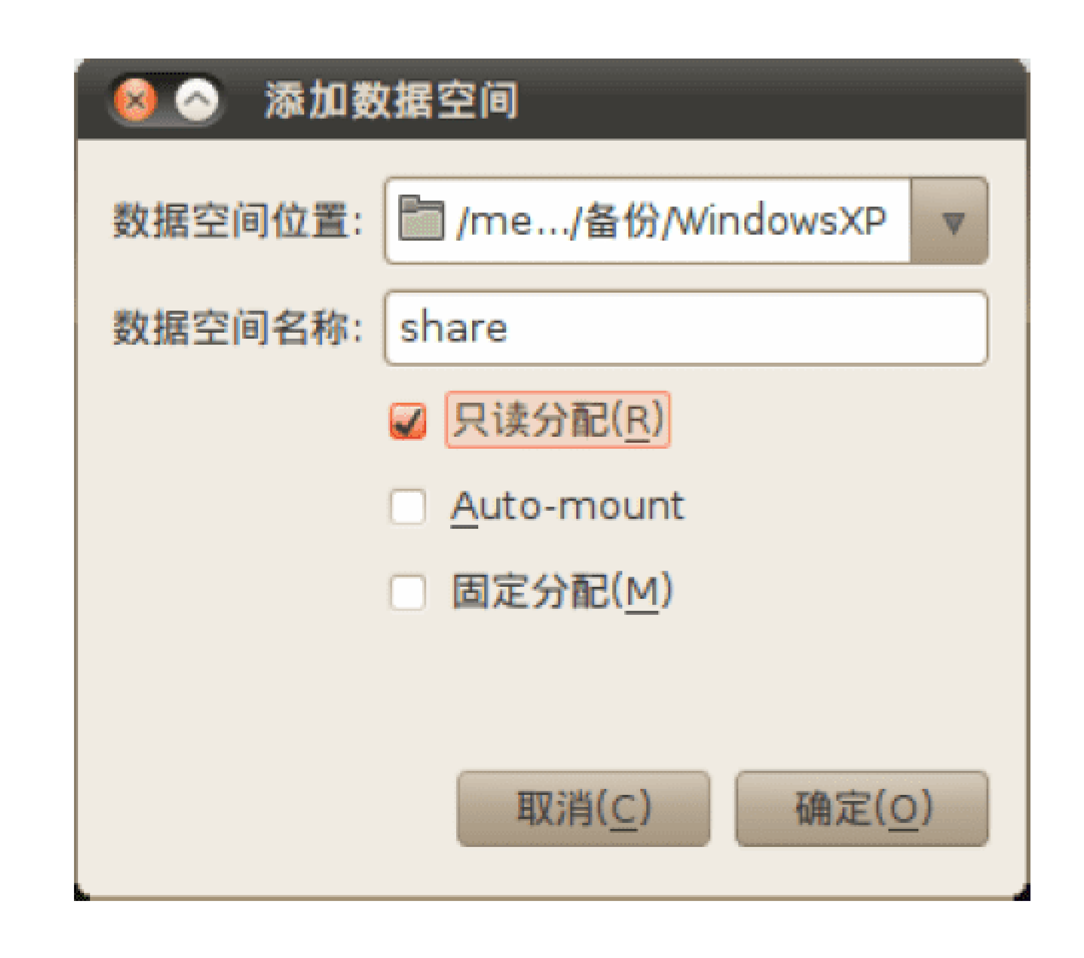




图 5.41　添加数据空间



图 5.42　创建了临时分配数据空间

（4）盒子妹的设置这就完成了，现在该去操作那个虚拟的 Windows XP 了。只见懒蜗牛右击了 Windows XP 的“我的电脑”，选择了“映射网络驱动器”，如图 5.43 所示。

（5）然后会弹出一个对话框，让懒蜗牛选择要映射的网络文件夹，如图 5.44 所示。懒蜗牛单击了“浏览”按钮。

（6）在弹出的“浏览文件夹”窗口中，懒蜗牛选择了“整个网络”|“VirtualBox Shared Folders”|“\\VBOXSVR\share”，如图 5.45 所示。



图 5.43　在虚拟机中映射网络驱动器

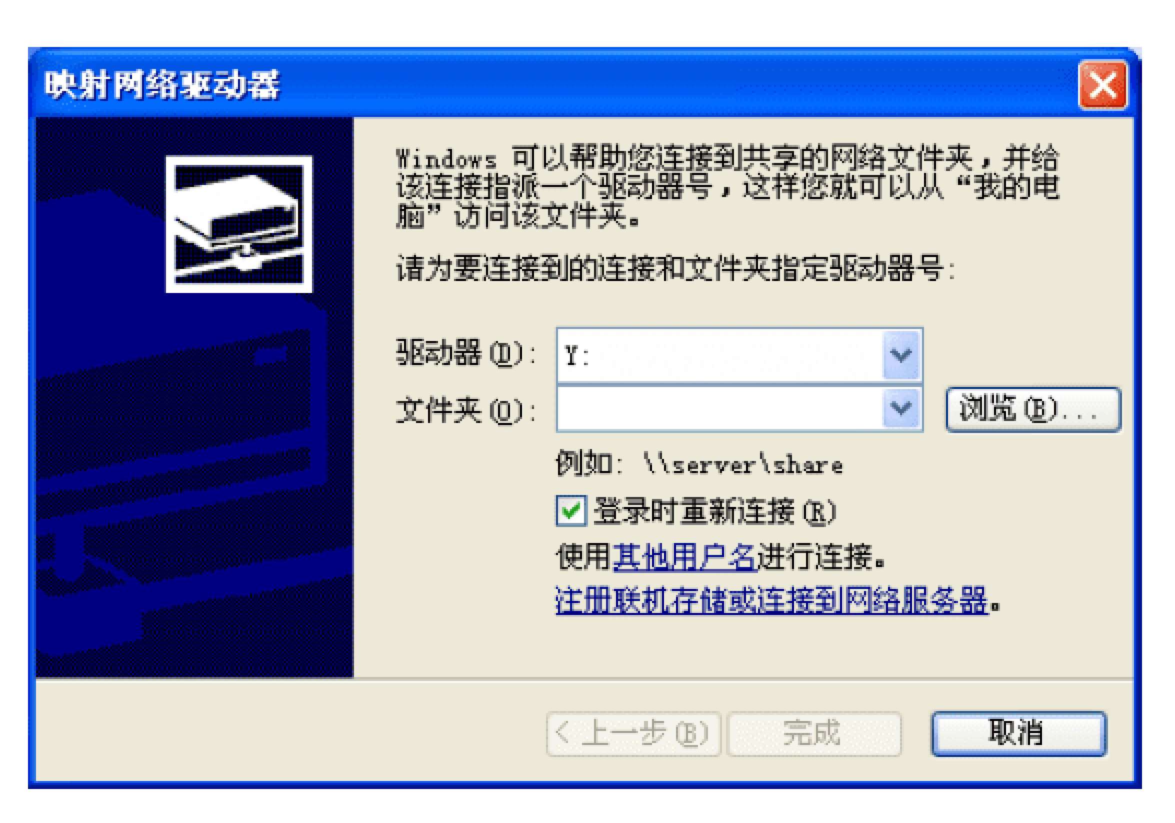




图 5.44　映射网络驱动器　　　　　　图 5.45　浏览文件夹

（7）选中之后单击“确定”按钮。这样，在这个虚 Windows XP 的“我的电脑”里面就多出了一个网络驱动器，就像图 5.46 所示的那样。双击进去，就是看到刚刚被懒蜗牛指定的那个目录的内容了，Office 的安装程序就在那里。



图 5.46　在“我的电脑”中查看刚刚创建的数据共享

这之后的过程就没有什么意思了。懒蜗牛在 Windows XP 中进入刚刚创建的那个空间，双击那个 setup.exe 文件，就开始了 Office 的安装。在一堆毫无悬念的“下一步”之后，就装好了。

5.2.7　更多虚拟机介绍

内存里的软件们逐渐熟悉了 Windows XP 的工作场景，渐渐地都把目光从大玻璃笼子里的 Windows XP 身上收回来，并且从“好奇”档切换到“崇拜”档，然后投向盒子妹。大家纷纷称赞：“盒子妹你真厉害！”“竟然能控制好这么大的内存，里面关着 Windows XP 还不让它溢出。”“你是怎么给 Windows XP 虚拟出那么一块显卡的？”

盒子妹被崇拜得不大好意思，向大家说道：“其实……也没有什么啦。我只是创造出一个虚拟的计算机而已。有这种能力的，也不是只有我有一个，像 Qemu，VMware 这些都是值得我学习的前辈。”

【老牌的 VMware】

gedit 小弟忙问：“VMware？这是啥？说来听听。”

狐狸接过来说：“我来介绍吧！VMware，这是个比较老牌的虚拟机软件，各方面都比较强悍。长相有些纠结，就是如图 5.47 所示这个样子。他的稳定性是很高的，好多企业服务器都用到他。”



图 5.47　VMware 的 Logo

我不解地问：“服务器用虚拟机干什么？”盒子妹过来解释：“主要是为了在节约硬件成本的同时，提高系统解决方案的安全性。打个比方吧，比如一个公司内部，要架设一个邮件服务器，和一台 ftp 文件共享服务器。由于用户不多，所以这两个服务都不需要很高的硬件配置。完全可以在一台服务器上跑这两个服务。”

“那就装俩呗！”gedit 忍不住说。

“但是，”狐狸妹妹接了过来，“如果提供邮件服务的软件有漏洞，就可能有入侵者通过漏洞获取整个系统的管理员权限，然后就可以影响到 ftp 服务。反过来也一样。于是这台服务器被攻破的概率就变大了。是吧，妹子？”

盒子妹点点头：“嗯！就是这个意思。在一个系统中，开启的服务越多，可能引来的漏洞就越多，一旦其中一个服务被攻击，就可能影响其他的服务。”

狐狸也点点头：“对，就是这意思。”

盒子妹继续说：“但是用 VMware 这样的虚拟机就不同了，可以在服务器上装虚拟机，再在每个虚拟机里分别装系统，每个虚拟系统分别跑一个服务。”

狐狸：“对，分散。”

盒子：“这样就算某个系统上的服务受到攻击……”

狐狸：“也不影响其他服务。”

盒子：“所以就等于把风险……”

狐狸：“分散了。”

盒子：“这样安全性就……”

狐狸：“提高了。”

我和 gedit 都点点头：“行，你们俩说相声去吧！”

【Virtual PC】

“那还有什么虚拟机软件呢？”gedit 继续问。

狐狸说道：“那时候还有个叫 Virtual PC 的，一开始称得上跟 VMware 并驾齐驱了。那时候的情况是，VMware 对内存的需求比较大，而 Virtual PC 对硬盘用得比较多。不过后来那个 Virtual PC 被微软公司收购了，自此以后就慢慢地销声匿迹了，我找到一张他工作时的照片，如图 5.48 所示。”众软件看了纷纷叹气摇头，惋惜这位明珠暗投的软件。

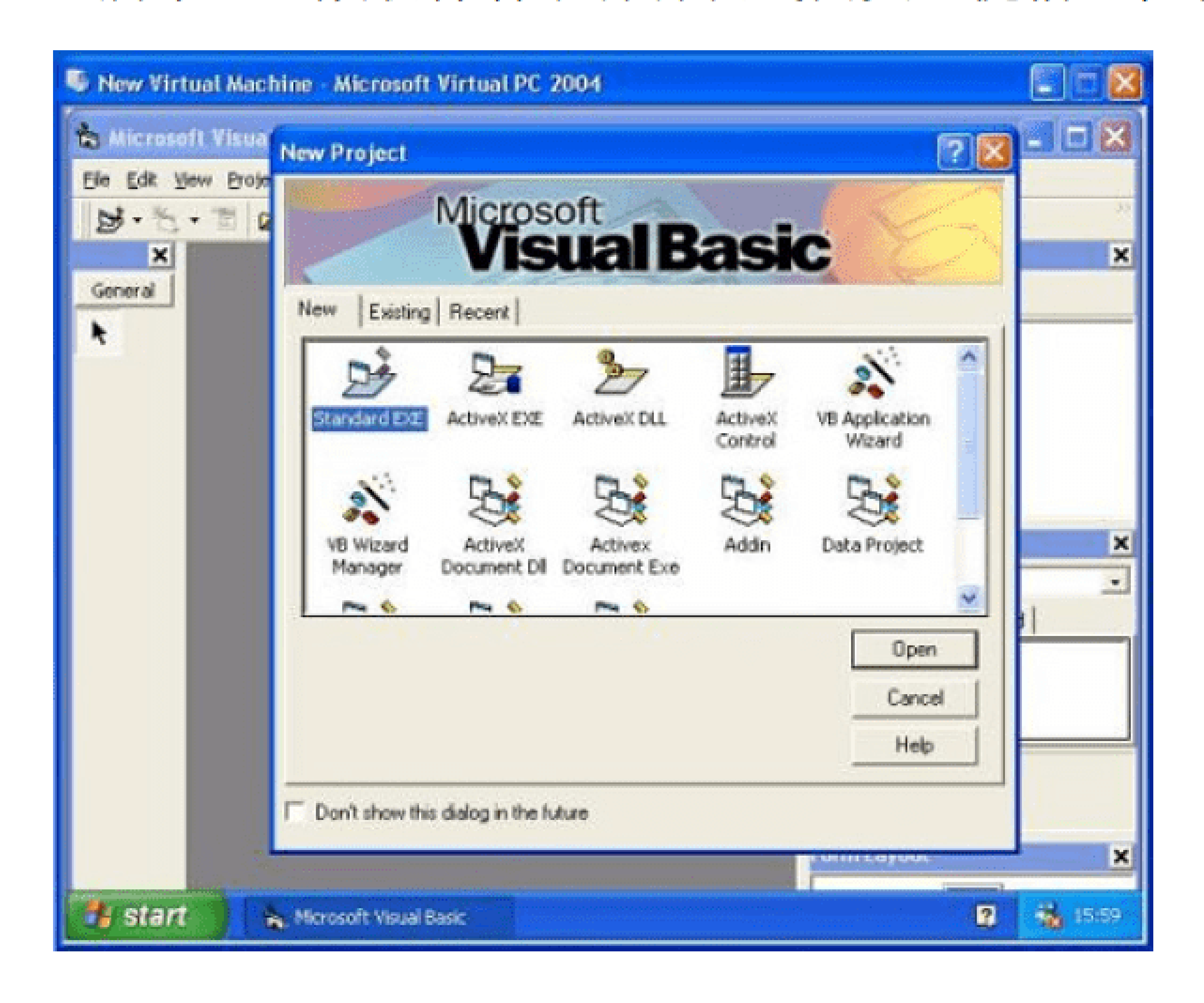


图 5.48　Virtual PC 中运行的 Windows XP

🔔提示：Virtual PC 被微软收购后，不再提供对 Linux 主系统及客户系统的支持。即：既不能在 Linux 系统中安装 Virtual PC 软件，也不能在 Virtual PC 虚拟的计算机内安装 Linux 系统。后来自 Virtual PC 2007 版本以后，再没有更新桌面版 Virtual PC。

【Qemu】

狐狸妹妹稍稍停顿后继续说：“后来，咱们开源界自然也不甘寂寞，有了 Qemu 这个虚拟机。这是一个字符界面的软件，使用起来有些费劲，不过功能很强大，可以和 GDB 协作，用于调试内核代码。也就是在 Qemu 创建的虚拟机里面跑一个内核，然后在虚拟机外开着一个 GDB 来进行调试。再加上他开源的性质，可以很方便地迁移到各种硬件平台，于是成为嵌入式开发者常用的软件。”

“哦……”大家虽然“哦”了一下，但我估计还有很多软件不知道嵌入式是什么意思。

狐狸妹妹继续说：“后来呢，Qemu 确实使用不方便，不过好在是开源的，于是就有人在 Qemu 的基础上进行改造，添加了图形界面，并且增强了对显卡的支持等，最终把 Qemu 改造成了一个很好用的虚拟机。”gedit 问：“改完了叫什么？还叫 Qemu 么？”狐狸微微笑了一下，望向盒子妹，只见盒子妹不好意思地说：“就是我啦，我是在 Qemu 的基础上修改而来的。”

【KVM】

继续听狐狸介绍：“再后来呢，出了那个叫 CPU 的虚拟技术，就是让 CPU 可以支持直接被虚拟机使用，比如 Intel 家的 VT-x 技术和 AMD 公司的 AMD-v 技术，虽然实现可能不大一样，不过从根本上讲其原理都差不多。”

gedit 问：“这种技术是干什么用的？”

盒子妹解释道：“主要就是可以提高虚拟机的运行效率。对于这种技术支持得最好的，

可以说就是 KVM 了。不过他需要内核的支持，并且本身并不是一个完整的虚拟机软件，因此需要依赖 Qemu 来运行。”

狐狸接过盒子的话说：“这样运行起来的虚拟机，运算效率大幅度提高。但是 KVM 虚拟的显卡很差劲，所以用来虚拟不需要太多显示任务的服务器应用很有意义，但是对于桌面应用，看片、玩游戏，就不如咱们的盒子妹妹了。”盒子妹又不好意思地说：“姐姐过奖了。”

提示：现在的 VirtualBox 同样可以支持 CPU 虚拟化技术。在虚拟机管理界面单击“设置”按钮，在左侧列表框里选择“系统”，然后在右侧选择“硬件加速”标签，如图 5.49 所示。勾选“启用 VT-x/AMD-V”，以及“启用 Nested Paging”复选框即可（前提是确认你的 CPU 支持相应虚拟化技术）。

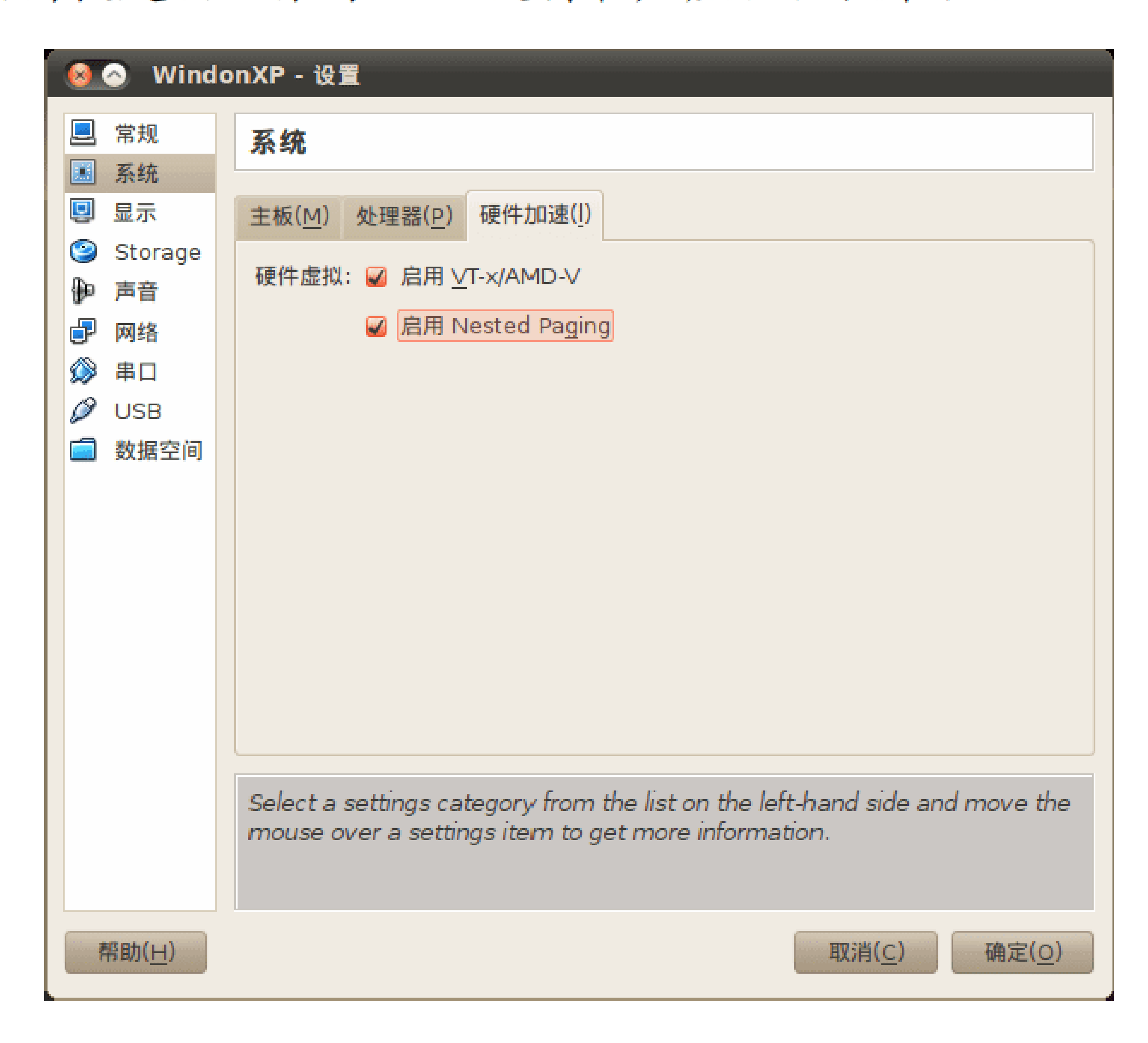


图 5.49　设置硬件加速

5.2.8　扩展阅读：虚拟化技术

刚才说到了 CPU 的虚拟化技术，顺道介绍介绍这个 CPU 的虚拟化到底是怎么回事吧。

【只有操作系统才能随意使用 CPU】

话说这个 CPU 可是我们程序要使用的重要设备，每个程序都离不开它。可是 CPU 很贵，不能发给每个程序一个（否则懒蜗牛同学会破产），于是就得由我统一管理 CPU 的使用。狐狸妹妹来了，我会把 CPU 给她用，gedit 小弟也来了，他也要用，那么我就告诉他俩，一人用一阵儿。

像狐狸，gedit 这些普通的程序我都可以管，因为我是内核嘛。普通的程序们也都知道我让他们停止使用 CPU 时，一定得停止。并且，CPU 的哪些功能是可以用的，哪些是不能用的，他们也都清楚。但是现在盒子妹饲养的 Windows XP 工作，就不一样了。

提示：某些 CPU 的高级命令只有内核级别的代码才能够使用。

【虚拟的操作系统也想随意使用 CPU】

Windows XP 在工作的时候，是不管其他人的。首先他根本也看不见其他人，再者，他也是个操作系统，正常来说，他运行在一台机器上的时候，所有软件都得听他的。他可以随便使用任何硬件，当然包括这个 CPU，就像我在我们这里的地位一样。然而当他运行在虚拟机里的时候，我们可不能真的让他随意使用 CPU，否则万一他执行个啥特权级指令，搞不好我们整个系统就挂了。

你问啥叫特权指令？简单地说，CPU 的指令分为两种，普通指令和特权指令。一般软件用 CPU 的时候都是执行各种普通指令，特权指令只有我这个操作系统可以执行。之前不是说 CPU 就像是个计算器么，特权指令就像这个计算器上的一些特殊的按键，实现一些特殊的功能，比如关机、复位什么的。这些按键一般人不许动，只有我可以按。

【让虚拟的系统用虚拟的 CPU 去吧】

那么既然有我在，当然就不许 Windows XP 动这些东西。可是盒子妹要给 Windows XP 营造一个真实的硬件环境才行，怎么办呢？很简单，虚拟一个假的给 Windows XP 用。看 Windows XP 给虚拟的 CPU 发了普通指令，就扭头向我申请使用 CPU，然后把那个指令在真实的 CPU 上执行一下，最后把结果传回给 Windows XP。如果 Windows XP 执行了一个特权命令，那么盒子就不真的去执行，而是模拟一下执行那个命令后应该有的效果。比如 Windows XP 执行了 CPU 自爆指令，盒子妹就在那里模拟：“轰隆！咔嚓，哎哟……”然后 Windows XP 就信以为真了。

【虚拟化技术让虚拟系统能用上真的 CPU】

那现在回来说这个 CPU 的虚拟化技术。

Windows XP 的每一个命令都由盒子妹转发是一件比较耗费时间的事情，于是人们提出了 CPU 的虚拟化技术。支持虚拟化技术的 CPU 有两种操作模式：VMX root operation（根虚拟化操作）模式和 VMX non-root operation（非根虚拟化操作）模式，统称为 VMX 操作模式。VMX root operation 就是平时我们用的模式。而 VMX non-root operation 则是像盒子妹饲养的 Windows XP 这样的虚拟机系统所用的模式，在这种模式下，一些特权指令可以执行，但不会对真机起任何作用。

有了这种技术后，盒子妹就轻松了。Windows XP 要用 CPU 的时候，不用虚拟一个 CPU 给他用，而是直接向我申请使用 CPU，然后把 CPU 开到非根虚拟化操作模式，之后直接扔给 Windows XP 使用就可以了，反正这个时候那个自爆按钮已经不管用了，扔给他随便玩去吧！

5.3　本 章 小 结

本章中，咱们的笨兔子可算是认识了 Windows 系统里的不少软件。谁让这些软件只有

Windows 的版本呢。懒蜗牛非得用这些软件，那就只能是模拟运行了。

要么模拟 Windows 的系统环境来运行软件，这就是 Wine 的工作；要么就是干脆模拟出一台计算机，里面装 Windows，这就是 VirtualBox 的工作。不过，模拟终归是模拟，偶尔应急还行，真正顺手的还是咱系统里的原生软件。不知道懒蜗牛同学还将体验到哪些有意思的原生软件，诸位，敬请期待下一章吧。

第 6 章　命令行的使用

虽然我们 Linux 的图形界面已经比较先进了，绝大多数操作都完全可以用图形界面来完成，但是就像吃过麦当劳肯德基不等于吃过西餐一样，连终端都没进去过，你也好意思说你会用 Linux？因此懒蜗牛同学决定开始学习 Linux 的命令行了。

6.1　这就是命令行

学习 Linux 命令行，离不开一大堆的命令。不过在这之前，首先要对命令行有些了解。下面就来说说命令行。

6.1.1　初识终端

命令行并不神秘，打开“应用程序”|“附件”|“终端”，你看到的就是，如图 6.1 所示



图 6.1　Gnome 终端

提示：也可使用快捷键 Ctrl+Alt+t 来打开终端。

这种图形界面下的命令行窗口，叫做伪终端。在这个窗口里，你可以近距离地跟我交流。我们操作系统是很希望用户能够和我们使用命令来交流的，像朋友般倾诉，感受彼此的心声，而且还低碳环保，节省能源（省 CPU 和内存啊）。

提示：如果只是想要通过命令运行一下某个软件，例如 gconf-edit，可以按下 Alt+F2 组合键并输入命令来执行。

【两种老板】

用户对于我们操作系统来说就相当于老板。使用图形界面的用户和使用字符界面的用户是两种完全不同的老板。

前者高高在上，拒人千里之外，就会比划，有事都不直说。比如他指着电灯开关的按钮冲你“嗯”一下，那就是让你过去把电灯打开；他用手一指桌面上的文件，冲你“嗯”一声，那是想让你选中这个文件；他要是指着文件冲你“嗯，嗯”两声，你就得明白他是让你打开文件（这里你还得注意，他“嗯”的那两声之间的间隔长短，要把握好分寸。要是间隔很短，那叫“双嗯”。要是间隔长，那叫“嗯”两下，对应的操作是不一样的）。他要是用手冲着桌面上的一堆东西指一圈，然后嘴里拉长声的“嗯～”一下，那就是让你把这些东西全都选中。给这种老板打工，需要很大忍耐力啊。

要是遇到第 2 种老板就要好得多，这种老板，平易近人，没有架子，有事会跟你平等地交流：“小笨啊，你列个表，看看我这目录下都有什么东西啊。”“小兔啊，3 分钟后下班啦，到时候记得关机啊。”“小笨啊，咱挺寂寞的，放个片子看看吧。就放那个 wall-E.avi 吧。”遇到这样的老板，工作起来，才是心旷神怡。

用命令行的用户，就相当于这第 2 种老板，跟他们交流，舒服。

【各种终端】

图 6.1 所示的那个终端，是 Gnome 环境中默认的伪终端软件，叫做“Gnome 终端”。所谓伪终端，是相对于那个脱离图形界面的、黑漆漆的、按 Alt+Ctrl+F1 组合键出来的那个终端来说的。伪终端的工作是在桌面上申请一个窗口，然后在里面模拟显示出一个黑漆漆的界面。

除了 Gnome 终端之外，还有很多其他的伪终端。像 konsole，这是 KDE 环境下的默认终端，如图 6.2 所示；xterm，是 xWindow 默认的终端，样子很简陋，什么也没有，如图 6.3 所示；还有 urxvt，这是一个简洁快速的终端，如图 6.4 所示。

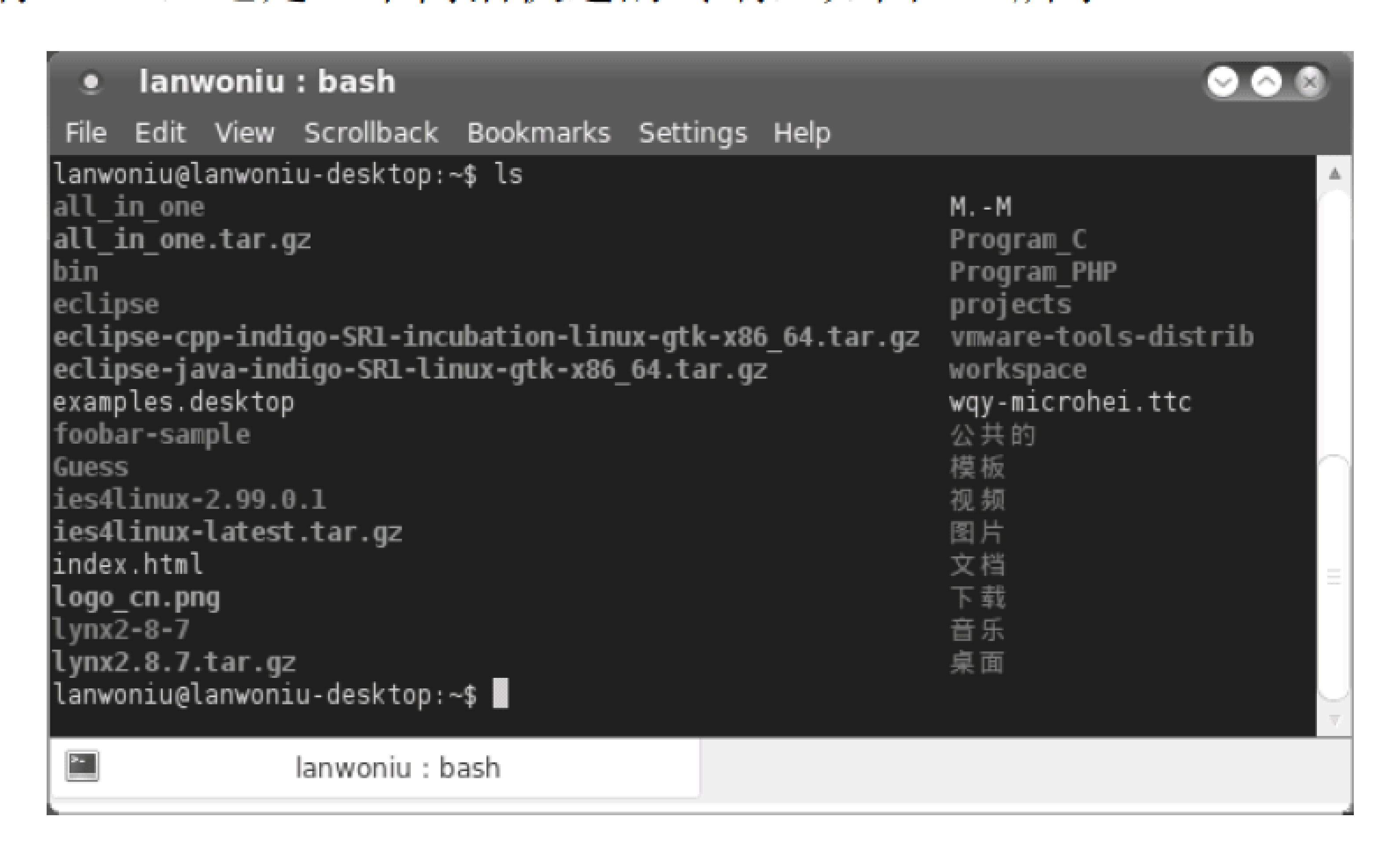


图 6.2　konsole 界面

除此之外还有很多的伪终端软件，你可以多装几个伪终端软件来对比一下，看哪个好

用。不过区别不太大，无外乎都是一些黑漆漆、白惨惨、绿幽幽的窗口，在里面可以输入命令而已。

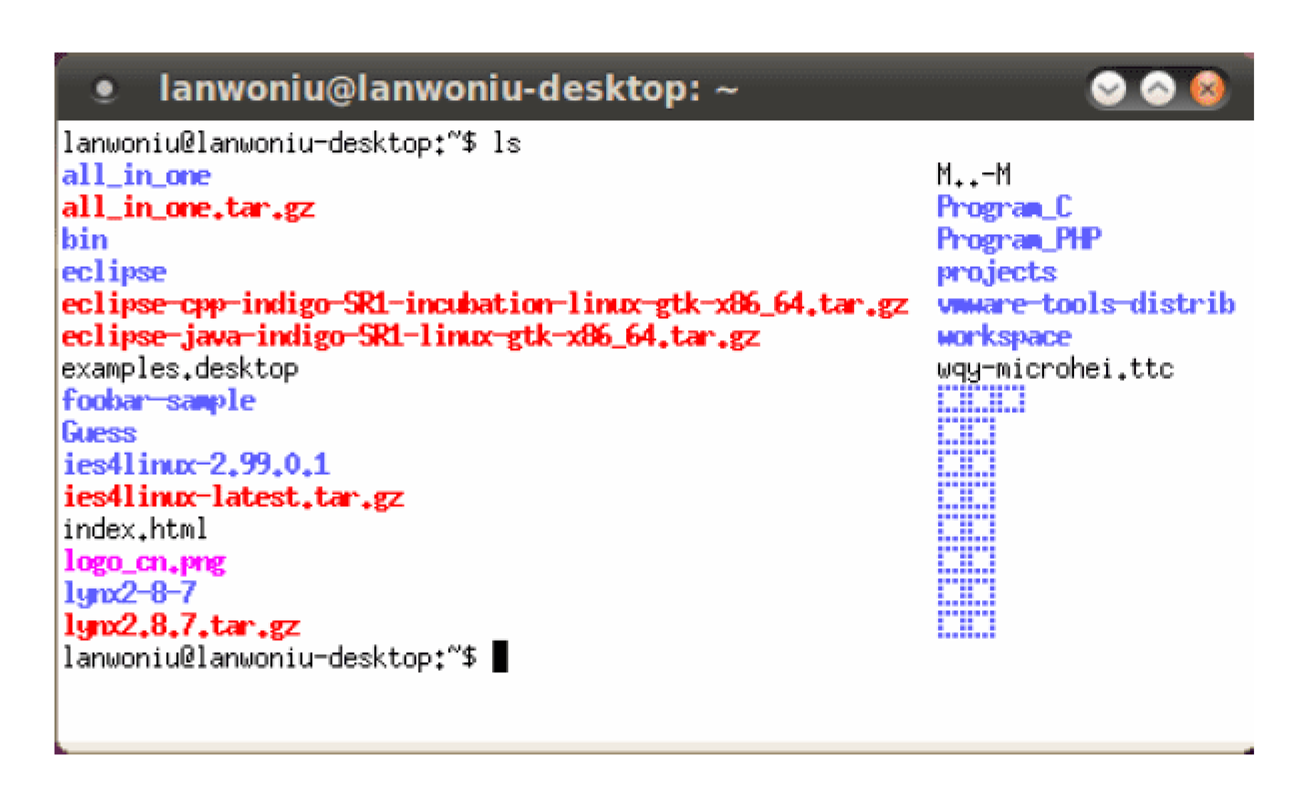


图 6.3　xterm 界面



图 6.4　urxvt 界面

【真假终端】

有人可能要问了：“你老说伪终端，那没图形界面，像 DOS 那样的命令行才是真终端咯？”其实也不是，那个我们管它叫做虚拟终端。那合着又虚又伪的，都不是真的。真的终端长什么样？

终端最初是一种输入/输出硬件设备，有键盘显示器和联接主机的接口电缆，是过去大中小型计算机上的概念。咱们用的 PC（也就是微型机）是没有终端这一概念的。当然我们 Linux 也可以通过串口外接字符终端，不过这年头有串口的 PC 也不多了。

虚拟终端就是把一台全功能计算机（比如咱们的 PC）通过软件模拟成另外一台计算机的终端设备，因此叫做虚拟终端。

有的人可能会有顾虑：伪终端，是不是里面执行的命令不是真正的 Linux 命令啊？这些不一样的虚拟终端对命令的执行结果都一样么？不一样？错！一样？还是错！它们根本就不会执行命令，它们只是个界面而已，都是浮云。真正处理命令的不是它们，是 Shell。

终端只是负责提供一个输入命令的交互界面而已，在里面运行的命令并不归终端软件去解析，而是找到专门的命令行的程序，这种程序一般称为 Shell。

6.1.2　Shell 的基本概念

那么 Shell 又是个什么东西呢？

【Shell 的作用】

Shell 是啥？是海鲜馆的扇贝？是汽车用的润滑油？都不是，他是一个外壳。什么叫外壳呢？咱们慢慢说。

我们 Linux 是个内核，这个内核是可以做很多事情的，整个电脑的硬件都归我管。显卡、声卡、内存、硬盘都归我控制；硬盘上的各种程序也归我调度。那么，我应该用这些硬件软件去干点什么事情呢？我不知道，因为没有人给我下命令。下命令的就是人类用户，比如我的懒蜗牛同学。

可是人类用户要做什么操作，靠他拿嘴说，我肯定是听不懂的。因此就需要一个能够把人类用户的操作意图转述给我的软件，这个软件就是 Shell。他就像罩在我这个操作系统和人类用户之间的一个外壳一样，在我和人类之间相互转达信息。

提示：Shell 广义上可以指操作系统和用户接口的界面，图形界面也是一种 Shell。因为图形界面的本质也是实现“把人类用户的操作意图转述给内核”。

【Shell 的种类】

Shell 有很多种，有 bash、csh、ksh 等，各有特点。

- bash——这是最常见的 Shell 了，全名为 Bourne Again Shell。基本上多数发行版都用他作为默认的 Shell 程序。他的各项功能都比较完善，是个全能型选手。
- csh——也就是 C Shell，之所以叫这个名字，是因为他的语法是按照 C 语言的风格设计的。人机交互的感受相对好些，但是语法相对复杂，并且执行效率不高。

提示：Linux 下的 csh 并不是原先 UNIX 系统中的 C Shell，而是一个 C Shell 在 Linux 下的免费的重实现版本，tc Shell。

- ksh——全名 Korn Shell，这也是个很厉害的 Shell，功能强大。不过 Bash 比他出生得晚，因此借鉴了他的大部分优点，于是 ksh 的使用也就越来越少了。

【Shell 的实质】

有的同学说了：说了这么半天，这个 Shell 到底是个啥软件啊，我怎么感觉不到他的存在呢？我从来没有运行过他呀。他跟终端有啥关系呢？

Shell，其实就是一个二进制的程序，跟狐狸、gedit 他们一样。只不过，Shell 的任务不是上网，也不是编辑文件，而是和用户交流。

比如我们 Ubuntu 系统中，默认的 Shell 是 bash，也就是/bin/bash 这个二进制文件。你没有亲自调用过他，是因为每次用户打开终端的时候，终端程序会自动调用用户的 Shell。

那么终端怎么知道用户的 Shell 程序是什么呢？这很简单，在/etc/passwd 文件里有记载。比如我们的 lanwoniu 用户打开了虚拟终端，G 终端就找到 passwd 文件里对应当前用

户的一行，类似下面这样：

```
lanwoniu:x:1000:1000:lanwoniu,,,:/home/lanwoniu:/bin/bash
```

这一行的最后一段就说明了这个用户的默认 Shell 是/bin/bash。于是 G 终端就去叫醒 bash，bash 起床，通过 G 终端来跟懒蜗牛用文字交流。那么 bash 最先说的一句话大概就是“你好”，当然不会这么不专业，这句话用 bash 的专业语言说出来就是下面这样：

```
lanwoniu@lanwoniu-ubuntu:~$
```

这一行是什么意思呢？

- “@”之前的，是当前用户的用户名。
- “@”后面，“:”前面是计算机名，这两个都好理解。
- “:”后面、“$”前面是当前所在目录，就是当前输入命令的人所在的位置。“~”代表用户的家目录，也就是“/home/<用户名>”这个位置。
- “$”则是命令提示符，在“$”后面就可以输入命令了。

提示： 普通用户的提示符是$，如果用 root 登录终端，则提示符是#。但 Ubuntu 系统默认禁用 root 用户，所以一般看不到#提示符。

6.1.3　bash 的工作（简单的 Shell 命令介绍）

正说着，懒蜗牛已经开始敲命令玩了。什么 ls、free、top、fdisk 等常用命令，挨个试验。于是工作间里也开始忙碌了起来。你可能以为 bash 会在懒蜗牛的指挥下跑来跑去，执行各种操作。其实完全不是那么回事，bash 只是作为一个命令的传达者而已，真正干活的是那些命令们，也就是 ls、free 这些家伙。

【bash 和图形界面的工作性质相同】

这些所谓的命令，其实都是一个个的小程序，或者说一个个的小软件而已。就跟狐狸妹妹、OO 老先生一样，只不过比他们小巧很多。如果你愿意，也可以把 Firefox 视为一个上网用的图形化界面的命令，为了方便描述，咱们以后管这些家伙叫做命令程序吧。

当用户输入命令比如 ls 的时候。ls 这两个字符就被传给了 bash。bash 怎么处理呢？首先 bash 要看输入的字符是不是自己的什么关键字，比如 for，history 之类的，如果是，就归 bash 来处理了；如果不是，就说明懒蜗牛是要找个命令程序，bash 就要负责去找到懒蜗牛想要的这个程序，并且叫他起床干活。

这个工作过程其实跟 Gnome 的工作是很相似的，只不过 Gnome 是根据鼠标的点击位置来判断用户想要运行哪个软件，而 bash 是根据用户输入的字符来判断的。

【bash 查找命令的艰辛历程】

那么 bash 去哪里找哪些命令程序呢？不知道您有没有听说过有个叫做环境变量的东西，跟 Windows 系统里的那个环境变量差不多，其中有个环境变量叫做 PATH，里面记录着 bash 去找程序的路径。如果你想看看 PATH 到底是什么，运行 echo $PATH 就可以了。会得到类似这样的输出：

```
lanwoniu@lanwoniu-ubuntu:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

提示：$符号在命令中表示引用变量。可以 export 设置变量（这里引用和设置的都是全局变量，后文会讲到）。例如：

```
lanwoniu@lanwoniu-ubuntu:~$export test="my test word"
lanwoniu@lanwoniu-ubuntu:~$echo $test
my_test_word
```

第 1 行设置 test 变量的值为字符串“my_test_word”，第 2 行使用 echo 命令来查看 test 变量，第 3 行是运行结果。

当懒蜗牛运行一个命令，比如 ls 时，bash 就对照着 PATH 里面的设置，开始找了。

他先去/usr/local/sbin 房间里面找（根据 PATH 的设置顺序）。敲开门，客气地问：“请问 ls 是住这屋么？”等了半天，除了墙角的蜘蛛网上那 8 条腿的小家伙寂寞地弹了几下琴弦之外，再也没有活物给他任何回应，于是 bash 意识到这屋没人，赶快去下一屋。

又到了/usr/local/bin，他依然是很礼貌地敲开门问候，里面只有一位懒蜗牛前几天安装的叫做 Maya 的软件，只听那位叽里咕噜地说了几句 2012 啥的玛雅语，bash 也听不懂，不过反正他不是 ls 就对了，赶紧去下一间。

提示：/usr/local 目录与/usr 目录中的结构类似，都包含 bin、sbin、lib、incud 等目录。对于 Linux 系统，并没有明确地对这两个目录内容进行定义。不过一般来说，对于 Ubuntu 系统，/usr 目录中用于存放从软件源中安装的软件。/usr/local 中用于存放用户用其他方式安装的软件。

bash 推开/usr/sbin 这屋的门一看，这回里面很热闹，而且都是重要人物。有管用户创建的 useradd、每天启动必备的 gdm、负责跟通过网络跟 Windows XP 共享文件的 smbd 和 nmbd 等。一听 bash 来找 ls，useradd 没好气地说：“哎呀，ls 怎么可能在我们这里呢？我们这里都是管理级的程序，都是领导！那个 ls 是谁都能运行的，他怎么会在 sbin 里，你得去 bin 里面找啊。”

bash 只好客气地退了出去，继续去找/usr/bin、/sbin/、/bin，终于在/bin 里面找到了 ls，于是赶紧叫醒 ls，让他去干活。至此，bash 的任务也就结束了，他就回去等待懒蜗牛的下一个命令了。

提示：一般/bin、/usr/bin、/usr/local/bin 目录下存放的是普通用户使用的命令。/sbin、/usr/sbin、/usr/local/sbin 目录下存放的是需要 root 权限才能使用的命令。

上面说的是直接敲一个命令的情况。直接输入一个命令，bash 就会去 PATH 变量记载的路目中查找。如果你想运行一个二进制程序（比如刚刚从网上下的某个软件的安装程序），但这个程序不在这些/bin、/sbin 之类的目录里怎么办呢？也好办，只要你运行的时候加上路径就可以了。如果没有写路径，那么 bash 就去 PATH 中的目录里找，如果写了路径，他就直接去指定的路径找了。

提示：Linux 中，一般当前目录——即“./”，并不在 PATH 变量中，所以要运行当前目录下的二进制文件或脚本文件，也需要加上路径。例如要运行当前目录下的 setup.sh 脚本，需要输入：./setup.sh。

【有困难找纯爷们儿】

懒蜗牛对照着书本练习各种命令。学到 ifconfig 命令的时候，觉得书上写得不是很明白，怎么办呢？bash 告诉他：“别着急，我给你找个人问问，这个人，纯爷们儿！”然后 bash 扭头冲着硬盘里喊：“嘿 man！你出来，说说这是怎么个意思。”

随着咔咔嚓嚓一阵硬盘响，只见内存中走来一人。见此人人高马大，膀大腰圆，扇子面的身材，胳膊跟大腿一样粗，这就是 bash 说的那个“man”。这纯爷们儿说出话来如同打个炸雷一样：“嘿！你好啊。洒家我是专职命令解说员，你有什么想知道的吗？”懒蜗牛输入了命令：

```
man ifconfig
```

意思就是问 man，这个 ifconfig 怎么用啊？man 仰天大笑一声：“嚯哈哈哈，要说这个 ifconfig 嘛……不难，听我慢慢地道来！”当然，他不是用中文说的，而是用英语介绍了一下 ifconfig 命令的使用方法，如图 6.5 所示。

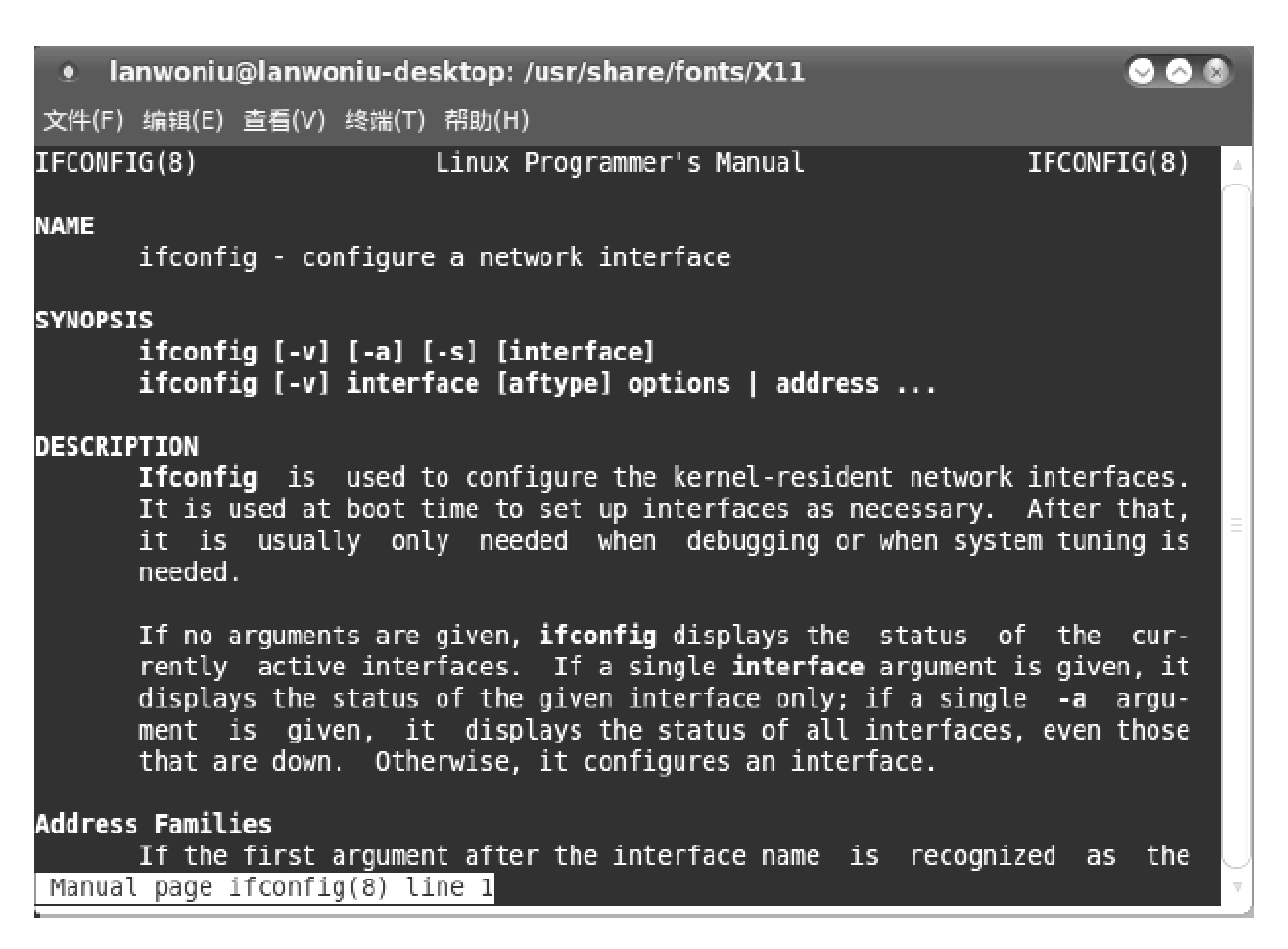


图 6.5　man 查看 ifconfig 命令

提示：进入 man 的界面后按 q 键可退出 man。

6.2　这么用 Shell

懒蜗牛同学那敲敲打打的命令行生活就这样开始了。从那以后，工作间里少见了红酒大师；看不到盒子妹妹；心有灵犀也不常来上班了；OO 老先生也难得起床了，工作间里净是些命令行的小程序在跑来跑去。

6.2.1　理解目录结构

这一天，懒蜗牛觉得玩伪终端不够过瘾，于是按下 Ctrl+Alt+F1 组合键进入黑漆漆的终端界面来敲名令。看着一屏屏的字符，懒蜗牛感觉很有成就感。

提示：进入终端后，可以按 Ctrl+Alt+F7 组合键回到图形界面。

【当前目录，家目录】

懒蜗牛进入终端后，先习惯性地运行了一下 ls，看到了当前目录下的所有文件。

所谓当前目录，就是用户现在所在的目录。比如你在你家卧室发呆，那么你的当前目录就是卧室；过一会儿你又去客厅发呆了，那么当前目录就是客厅；然后你去厕所发呆，当前目录就是厕所（怎么到哪都发呆）。那么懒蜗牛现在的当前目录是哪个目录呢？就是他的家目录，就是图形界面的“位置”下的“主文件夹”那个目录，也就是/home/lanwoniu 这个目录。当懒蜗牛每次打开终端的时候，无论是虚拟终端还是按 Ctrl+Alt+F1 组合键进入的终端，刚一进去，都是在懒蜗牛的这个家目录里。

提示：命令行下可以用“~”符号代表当前用户的家目录。

【出去走走】

不过，毕竟不是什么事情都要在家目录里做的，如果你在这个目录里看够了，想出去走走，到其他的目录逛逛，这时候就需要 cd 命令了。cd 命令跟光盘没有关系，他是 Change Directory（改变目录）的缩写。这个命令可以改变当前的目录，他就像出租车一样，可以让你到达你想去的任何一个目录（当然，前提是你得有权限进入那个目录，就像你不能让出租车开进中南海一样）。

cd 命令的用法就是：

```
$cd <路径>
```

【绝对路径和相对路径】

那么，路径该怎么写？如何描述你想去的目录呢？一般有两种方法：绝对路径和相对路径。

绝对路径就是无论去哪都统一从一个根本的位置上描述。比如你打车，司机师傅问你去哪，你说：“我去地球，亚洲，中国，北京市，崇文区，羊肉大街，排骨胡同，376 号。”这么说就是绝对路径。无论你在什么地方打车，都从一个根本的位置上说起（比如地球，当然，你愿意从太阳系说也行）。层层递进，最终说到最小、最详细的那个地方为止，就肯定错不了。我们这里目录的地址当然不能从地球开始说了，我们的根目录“/”就是那个最初的、根本的位置，无论你去哪个目录，都可以从这里说起，比如：

```
$cd /usr/share/fonts/X11
```

这条命令的意思就是说，要去根目录下的，usr 目录下的，share 目录下的，fonts 目录下的，X11 目录。

绝对路径很准确并且最直接，不过有时候也比较费劲，所以，cd 还支持相对路径。相对路径，就相当于你打车，司机师傅问你去哪，你说：“就前面那路口，左转，过三个红绿灯走 700 米有一家精神病院，到时候我指给您，您就靠边停车就行了。”这种描述方法就是以当前所在的地点为起始地点进行描述，而不用从外太空开始说。那么具体到我们这个系统里怎么说呢？还拿刚才那个命令作为例子。假如现在已经在/usr/share 目录下了，那么就运行：

```
$cd fonts/X11
```

这样就进入 X11 目录了。这句命令的意思就是，要去当前目录下的，fonts 目录下的，

X11 目录。

【引导员】

懒蜗牛同学学会了 cd 命令，兴奋地在终端里 cd 来，cd 去的，像个跑进游乐园的小孩子。转着转着，忽然觉得有点迷路了。静下来想一想：我现在是在哪个目录呢？是在/usr/bin，还是/bin，还是/usr/local/bin 呢？

其实，他看一眼那个提示符$前面的内容就可以了，默认设置下这里显示的就是用户所在的目录的绝对路径。不过这个提示符的格式是可以修改的，如果修改了，这里显示的不是当前的路径了，怎么办呢？

这一点，我们早为您想到了。命令行中配有专业的引导员，告诉您您现在所在的位置，这位引导员就是 pwd。可别一看名字就以为他是负责修改密码的，其实他跟 password 一点关系都没有，他是 Print Working Directory 的缩写。用法简单，输入 pwd 就行了，他就会告诉你现在所在的目录。

6.2.2　重要的 TAB——命令补全功能

学习完 ifconfig 命令，懒蜗牛同学又开始研究 fdisk，这是个磁盘分区命令。学习了这个命令之后，懒蜗牛又学习了查看网络端口的 netstat 命令。渐渐地，懒蜗牛发现了一个问题：随着命令字符越来越多，敲起来越来越费劲了。

一开始 ls、cd、top 这样的命令很简短，输入也没觉得麻烦。可遇到字母多的命令，一遍一遍地敲就慢多了，有没有什么省事的办法呢？

【便捷高效的键盘】

很多人不喜欢键盘，不喜欢打字。其实想想，早在电脑刚刚被发明出来的时候，键盘就已经是每一台电脑所必备的输入设备。作为从那个字符界面的时代走过来的 Linux 系统，我们自然充分考虑通过键盘操作整个系统的便捷和效率问题。直到现在，使用键盘操作 Linux 都会拥有意想不到的高效率和成就感。

我以前很不明白，键盘可以发送上百个命令，用起来应该很方便才对，为什么人类就那么喜欢那个只能发送：上、下、左、右、左键、右键、滚轮这么几个命令的鼠标呢？（当然，有的鼠标还有一些额外的功能键，但是那也比键盘少啊）。后来见多识广的 OO 老先生给我解释，我才明白。原来是因为人类记忆力不行，没有我们软件这么可靠，记不住那么多个键，于是只好用那只能发送几个命令的鼠标了。

好了，绕得有点远，其实说起来在我们的命令行里通过键盘敲命令是很方便的，只是很多人不大熟悉如何节省时间而已，都以为用键盘和我交流跟用键盘和那个 DOS 系统交流一样麻烦呢。其实我已经很人性化了，就因为键盘上有个键——Tab。

【重要的 Tab 键】

看一个人的键盘，就可以猜测出他平时用电脑干什么。如果 W, A, S, D, U, I, J, K 这些键严重磨损，说明这哥们儿玩拳皇的；如果 A, Shift, Ctrl, 1, 2, 3, 4, …, 9, 0 键严重磨损，说明是个玩即时战略的，星际魔兽之类；如果 Alt, S 或 Ctrl, Enter 键磨损，大概是天天聊 QQ；如果 Tab 键严重磨损，那估计就是个 Linux 高手了。因为在 Linux 的命令行下，Tab 键起着命令补全的重要作用。

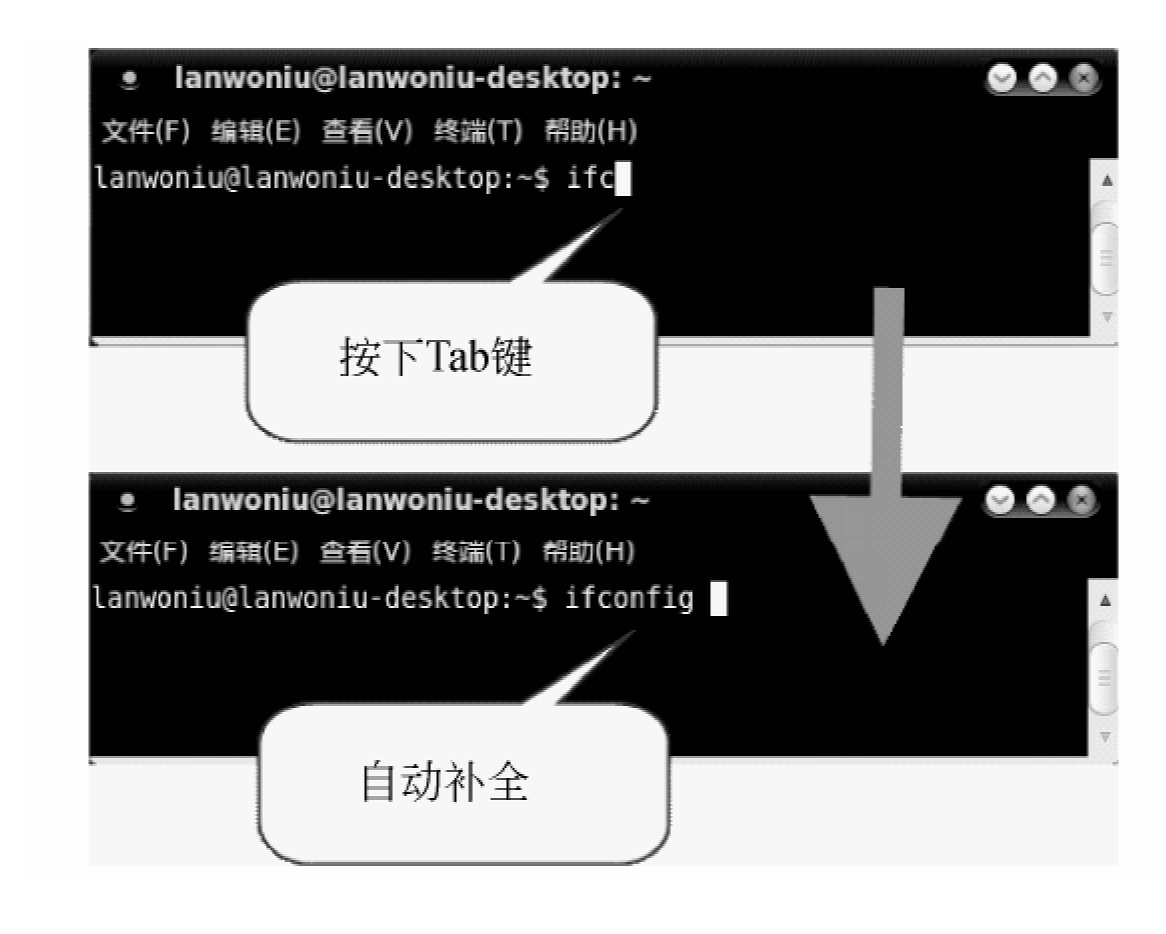
比如说，要运行 ifconfig 命令，可以不用完全输入这 8 个字母，只要输入 ifc，然后按

Tab 键，bash 就知道你要干什么了。因为所有可以运行的命令里面以 ifc 开头的就只有 ifconfig，所以当你按下 Tab 键的时候，它就会替你写出完整的命令：ifconfig，如图 6.6 所示。

这是因为在你按下 Tab 键的时候，bash 会去 PATH 变量所设置的所有目录里遍历一遍，检查了里面所有的有执行权限的文件，查到了 ifconfig 文件（命令其实就是个可执行文件嘛）。之所以这么快，是因为他早就把这些重要的东西缓冲进内存了，所以下次别抱怨我们 Linux 动不动就把你内存占满了哦。

提示：Linux 系统的内存管理机制是尽量多地使用内存。将空闲的内存用来做缓存。因此 bash 遍历 PATH 中的路径时并不是去硬盘读取，而是直接在内存中处理。

那么如果你再少写个字母呢？比如你只写了 if，然后就按 Tab 键，Bash 遍历了一遍 PATH 中的路径后发现，有 4 个命令是以 if 开头的，所以他不知道你要的是哪个命令，于是就不做任何动作。这时候如果你再按一下 Tab，他就会提示你：以 if 开头的命令有 if、ifconfig、ifup、ifdown、ifquery。然后你自己看需要的是哪个，照着输入就行了，很交互吧？如果没明白，可以看图 6.7 所示的效果。



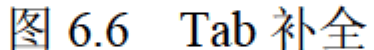


图 6.6 Tab 补全



图 6.7 Tab 提示

这样除了减少按键次数以外，还有一个好处就是你可以不必完全记住整个命令，能够记住前几个字母就可以通过 Tab 把整个命令回忆出来。

6.2.3 翻旧账——命令的 history

有了 Tab，用户输入新命令的时候省事了不少，还有一个 history 功能，可以让用户重复以前输入过的命令的时候省心。

在终端里，如果你想输入上一次输入的命令，按一下向上箭头，就看到了；如果想要再上一次的命令，就再按一下；如果想要再再上一次的命令，就再再按一下；如果想要再再再再上一次的命令……不怕键盘坏掉你就按吧。

好了，总该有些更靠谱的方法吧。如果想查看很久远以前的命令怎么办？请输入 history。这是一个命令，可以显示之前运行过的 *n* 条命令，默认情况下 *n*=1000。现在图形界面越来越发达，输入命令的机会越来越少，估计 1000 条记录都能把去年的命令显示出

来了。

🔔**提示：**也可以指定显示最近的 *n* 条命令。例如运行：

```
$history 20
```

显示最近的 20 条命令。

说起来 history 命令也没啥神奇的，他之所以能够显示曾经运行过的命令，不是因为他有啥水晶球，而是负责接收用户命令的 bash 会把每一条命令记录下来，写在~/.bash_history 文件中。用户输入 history 的时候，bash 再把这个文件打开，显示出里面的内容。

🔔**提示：**history 并不像 ls 那样是独立的命令，而是 bash 的关键字。即 history 是否可用与所用的 Shell 有关，本文所述仅针对 bash。

懒蜗牛输入了 fdi 这 3 个字母，然后按了一下 Tab 键，bash 赶紧给他把命令补全，fdi 变成了 fdisk，懒蜗牛同学感觉省事了不少（其实就少输入了俩字母）。听说这个 fdisk 是用来给硬盘分区的命令，可不能瞎玩，还是先问问纯爷们儿吧。于是，一阵嚯哈哈哈的笑声又从硬盘里传来了。

🔔**提示：**fdisk 需要操作硬盘设备，因此需要加上 sudo 提升权限才能运行。

6.2.4　more or less——命令的分页显示

懒蜗牛学习到查看当前进程，运行了一下：

```
$ps -A
```

这个命令会列出当前运行着的所有进程。以前在图形界面的伪终端中，懒蜗牛也运行过这个命令，列出了好多进程，一屏显示不下，滚屏滚上去了。懒蜗牛当时拖着伪中端的滚动条看了一下，这个动作如此自然，没有任何异样的感觉。然而这回他是在黑漆漆的、没有图形界面的终端里运行的这个命令，懒蜗牛同学发现了一个严重的问题——滚上去的信息怎么看啊？没有滚动条啊！有木有滚动条啊！有木有！！

【more】

好，我们让懒蜗牛同学自己咆哮一会儿去，咆哮的蜗牛我伤不起啊。不就是想向上滚屏嘛，很简单，有快捷键，Shift+PageUP 组合键是向上翻页，Shift+PageDown 组合键是向下翻页，就这样。

当然，可能光滚屏还不够，有时候想把命令输出的东西翻来覆去慢慢看，有什么办法没？办法肯定是有的，当初的前辈们都是生活在命令行下的。他们没事编译个软件，运行个测试脚本，输出的提示动不动就好几百行，一屏根本显示不下，除非用的是 32 寸宽屏液晶显示器，还得竖着放。

如果显示不下了，后面的内容就会把前面的内容“顶”上去。“楼主”固然是看不见了，什么“沙发”、“板凳”、“地板”的也一样没希望，只能看到最后那几十条。那么想看前面的内容怎么办？虽然可以用 Shift+PageUP 组合键来向上翻页，但也麻烦。一般人都是习惯从上往下看的，倒着往上翻就别扭了。那么怎么办呢？这时候 more 就该出场了。

提示：使用 Shift+PageUP 组合键向上翻看的页数受缓冲区大小的限制。修改缓冲区大小需要重新编译内核，此处不做讨论。

more 是一个用来阅读文本的命令行软件，他有分页显示的功能。他可以从文件或数据流中获取数据，把所有要输出的内容先显示出一屏来，等着用户按回车键，之后再显示第 2 屏，直到显示完全部内容。当然，用户也可以不等显示完全部就中途按 q 键退出。

那么具体怎么用呢？就比如刚才懒蜗牛同学这种情况，运行下面的命令：

```
$ps -A|more
```

其中有个“|”符号，这个符号叫做管道符，就在你键盘上的“\”键处。这个符号的意思就是把前面一个命令的输出内容交给后面一个命令作为数据输入。于是，ps -A 命令输出的那些字符就像一大堆泔水一样哗啦啦地都流到 more 这里了，more 就把这些东西先整个大桶缓存起来，然后盛出一碗来给你看：“您看有没有想吃的？”，你看完了之后，他再去盛第 2 碗，第 3 碗……直到你把一大桶泔水都检阅完了，more 才结束工作（当然，你要是坚持不到最后就吐了，记得按 q 键退出）。

有了 more 就满足了么？不，还有他的死对头，less。

【less】

less 实现的功能和 more 基本一样，也是用来分屏输出的，同行是冤家嘛。不同的是，less 比 more 强大，因此，在我们这里流行着这样一句逻辑混乱的话：less is more than more。

more 只能一页一页往下看，看完了就退出，不能向上翻。less 可以上下翻页，看过去的东西可以按向上键或者 PageUp 键翻回去看，比 more 更人性化一点。另外，都看完了之后 less 是不会自动退出的，一定要按 q 键退出。

提示：less 和 more 都可以直接查看文件，运行命令:

```
more /<路径>/文件名
```

或者

```
less /<路径>/文件名
```

就可以查看文件内容，前提是该文件是一个文本文件。

【只做一个功能，但要做到最好】

通过这两个命令，您大概可以感受到我们 Linux 系统和微软公司的系统的不同理念。用过 DOS 系统的都知道里面有个 dir 命令，和我们的 ls 一样，都是显示文件用的。当文件很多的时候，dir 命令有专门的参数可以实现分屏显示，而 ls 命令就没有，只能一下子显示出来。为什么？因为分屏显示的事情是由 more 和 less 负责的，完全可以通过 ls|more 这样的组合实现分屏显示的功能。

提示：Ubuntu 系统中也有 dir 命令，与 ls 命令完全一样，仅为照顾用户习惯。

Linux 软件的设计理念是“只做一个功能，但要做到最好”。每个程序只专注于一种功能的实现，而通过多个程序的组合可以实现任何功能。

试想如果没有 more 和 less 命令，ls 就要有负责分屏显示的功能。那么 history 命令是

不是也要处理分屏显示的问题呢？和 ps 命令一样，也得处理分屏显示的问题。那么所有输出行比较多的命令都要自己负责分屏显示，这些命令的源代码中就都要有负责分屏显示的部分，这是一种无谓的重复劳动。而且各自分别实现分屏显示，效果多半也不一样（因为这些命令多半不是同一个人写的），可能有的命令是按空格显示下一行，有的是按 n 键显示下一行等。与其这样，不如把相同的功能独立出来，成为一个统一的、单独的命令。

好了，说了这么多，这回懒蜗牛也终于结束了咆哮，回到图形界面找狐狸妹妹去查怎么在命令行下翻页了。

6.2.5　通配符

说起命令行下省事的办法，还有个东西不能不提，就是通配符。

【DOS 的通配符处理】

过去用过 DOS 系统的同学可能知道，通配符就是“*”和“?”两个符号。“*”号可以代表任意多个任意的字符，“?”号代表任意一个字符。不过 DOS 系统下通配符的实现是需要每一个命令对通配符都理解的。

比如说用户在 DOS 系统里运行：

```
copy *.jpg  aaa
```

这个命令的意图是，把当前目录下所有以.jpg 结尾的文件复制到 aaa 目录中。那么这个命令 DOS 系统是怎么处理的呢？DOS 这个懒得要死的家伙肯定啥也不管，只把 copy 叫醒，然后告诉他：“用户让我告诉你复制‘*.jpg’到‘aaa’，快干活去吧！”他根本不管*.jpg 是什么意思，只能由 copy 去理解：*.jpg 就是所有的以.jpg 为扩展名的文件，所以用户给他的任务就是复制（因为他只会复制，别的任务不会叫他来）当前目录下的所有.jpg 文件，以及.JPG 文件，以及.Jpg 文件，以及.jPg 文件，以及……（谁让 DOS 不区分大小写呢）复制到当前目录下的 aaa 目录去。

这样做有什么不好呢？这跟刚才分屏显示的问题一样，每一个可能需要支持通配符的命令都要自己实现对通配符的处理。重复开发，浪费资源。

【Linux 的通配符处理】

我们 Linux 下的 Shell 们就不这么懒了。就拿 bash 来说吧，也一样支持通配符，也是用“*”代表任意个任意字符，用“?”代表某一个任意字符。不过，通配符的解释都是由 bash 来做的。

还比如复制文件，用户运行：

```
$cp *.jpg ./aaa
```

这条命令输入进去之后，先交到 bash 这里，bash 一看有通配符，就需要自己进行解释。“*”代表任意长个任意字符嘛，那么把通配符进行替代之后，得到真正要执行的命令其实是（这里假设当前目录下有 11111.jpg、22222.jpg、33333.jpg、44444.jpg 这 4 个以.jpg 结尾的文件）：

```
$cp 11111.jpg 22222.jpg 33333.jpg 44444.jpg ./aaa
```

也就是说，bash 用当前目录下的所有 JPG 文件的文件名代替了*.jpg 这个字段，然后

再叫醒 cp，把扩展后的参数传给他。cp 看到的就是没有星号的命令了，他接收到的就是明确的命令："复制当前目录下的 11111.jpg、22222.jpg、33333.jpg、44444.jpg 这 4 个文件，到当前目录下的 aaa 目录下。"这样做的好处是明显的：bash 下的各种软件都不需要自己处理通配符的问题，减少了不必要的重复开发。

当然，也有的时候是不需要 bash 对特殊符号进行扩展的。比如懒蜗牛可能就有个文件叫做"*.jpg"，就叫这么个特殊的名字。要想复制这个文件而不是所有的 jpg 文件，该怎么办呢？也好办，就这样：

```
$cp \*.jpg ./aaa
```

在"*"前面加上一个"\"就可以了，"\"的意思就是后面的字符是个普通字符，告诉 bash 不要进行任何处理。

6.3 Shell 编程

懒蜗牛同学经过一段时间的学习，已经对我们 Ubuntu 系统里面的基本命令了如指掌了。接下来他又想要干点什么更有意思的事情呢？

6.3.1 把命令打包执行

【日复一日】

这一日，懒蜗牛把一张 SD 卡插进了电脑，然后运行 mount 命令来挂载：

```
$sudo mount /dev/sdb1 /mnt/
```

挂载之后，把里面的东西复制了进来：

```
$cp /mnt/*.JPG /home/lanwoniu/Picture/
```

之后把这些图片打成压缩包，放到一个目录里作为备份：

```
$tar -czvf /home/lanwoniu/Backup/PIC20101001.tar.gz /mnt/*.JPG
```

备份完了，把卡里的文件删掉吧：

```
$rm /mnt/*.JPG
```

最后，卸载这个 SD 卡：

```
$sudo umount /mnt/
```

又一日，懒蜗牛又把这张 SD 卡插进来了，里面有了新的文件。于是，他像前一次一样操作，又一次挂载：

```
$sudo mount /dev/sdb1 /mnt/
```

又一次复制：

```
$cp /mnt/*.JPG /home/lanwoniu/Picture/
```

又一次备份，又一次删除，又一次卸载：

```
$tar -czvf /home/lanwoniu/Backup/PIC20101002.tar.gz /mnt/*.JPG
$rm /mnt/*.JPG
$sudo umount /mnt/
```

然后第 3 天……懒蜗牛终于忍不住了。每次都要输入这么多命令很麻烦啊，能不能省事一点呢？

【高级批处理】

懒蜗牛同学回忆起了在很久很久以前，用过一个叫做 DOS 的操作系统，那里面有一种叫做“批处理”的东西，可以把很多条命令写进一个.bat 文件里，一起执行，似乎很强大。那么我们 Linux 系统里有没有这种批处理呢？在我们 Linux 系统面前说批处理，那简直就是关公面前耍大刀，华佗门口卖止疼膏了。

您或许听说过我们 Linux 系统中有叫做 Shell 脚本的东西。这是我们 Linux 的骄傲，如果批处理文件是辆自行车的话，Shell 脚本就是波音 747！这么强大的东西，解决懒蜗牛同学现在遇到的问题，绰绰有余。

要写个 Shell 脚本很简单，随便用一个什么文本编辑器，写上你要执行的命令，然后保存，就可以了。比如要解决懒蜗牛同学每次敲很多相同命令的烦恼，那么可以写这么一个文本文件：

```
sudo mount /dev/sdb1 /mnt/
cp /mnt/*.JPG /home/lanwoniu/Picture/
tar -czvf /home/lanwoniu/Backup/PIC.tar.gz /mnt/*.JPG
rm /mnt/*.JPG
sudo umount /mnt/
```

然后保存，随便起个名字就可以，比如叫“daily_backup.sh”。到此为止这个脚本还不能运行，还得赋予这个文件“可执行”权限。就这么操作：

```
$chmod +x ./daily_backup.sh
```

这就可以了。这回这个脚本就可以运行了，就这样：

```
$./daily_backup.sh
```

不过因为里面涉及挂载操作，所以脚本里面调用了 sudo。因此运行的时候会提示输入密码。这样，用这么一个脚本就可以省去懒蜗牛同学每次敲命令的烦恼了。

提示：脚本文件并不要求特定的扩展名，只要是文本文件，具有可执行权限即可。但一般习惯上将脚本文件的扩展名命名为.sh。

【灵活的 Shell 脚本】

不过有的同学可能发现了，这样简单地把静态的命令写成脚本，并不能完全解放懒蜗牛同学。懒蜗牛每天打包备份的文件都是不一样的，但是这个脚本里，备份文件的文件名是固定的呀，这样懒蜗牛同学每次运行完这个脚本还得去改一下文件名。就不能送佛送到西，帮忙帮到底么？

当然能，咱们来把脚本修改一下：

```
sudo mount /dev/sdb1 /mnt/
cp /mnt/*.JPG /home/lanwoniu/Picture/
```

```
tar -czvf /home/lanwoniu/Backup/PIC`date %Y%m%d`.tar.gz /mnt/*.JPG
rm /mnt/*.JPG
sudo umount /mnt/
```

这回，脚本中使用了“`”符号，“`”符号的意思就是：先执行两个“`”符号之间的命令，用执行后的输出代替掉两个“`”符号之间的内容（包括“`”本身）。

提示：要注意，这里的“`”符号不是单引号“'”，而是键盘上和“~”位于同一个键的反引号。

这里两个反引号“`”之间的“date”，就是用来打印出当前日期的命令。你可以在命令行下试着运行一下这个命令看看效果：

```
$date %Y%m%d
20111017
```

因此，如果在 2011 年 10 月 17 日执行上面那一段脚本，就相当于执行了：

```
sudo mount /dev/sdb1 /mnt/
cp /mnt/*.JPG /home/lanwoniu/Picture/
tar -czvf /home/lanwoniu/Backup/PIC20111017.tar.gz /mnt/*.JPG
rm /mnt/*.JPG
sudo umount /mnt/
```

于是，这样修改之后，这个脚本就可以根据运行当天的日期自动为备份出来的文件命名了。

提示：date 命令用于显示当前的日期及时间。直接运行 date 命令可得到类似如下格式的显示：

```
$date
2011 年 10 月 29 日 星期六 13:44:25 CST
```

或者也可在 date 命令后添加参数，以设定输出格式。可支持的格式参数如表 6.1 所示。

表 6.1 date命令参数

参　数	说　明
%H	小时（从 00～23）
%I	小时（从 01～12）
%M	分（00～59）
%p	显示出 AM 或 PM
%r	时间（hh:mm:ss）
%S	秒（00..59）
%T	时间（24 小时制）（hh:mm:ss）
%X	显示时间的格式（%H:%M:%S）
%Z	时区
%a	星期几的简称（例如 Sun）
%A	星期几的全称（例如 Sunday）
%b	月的简称（例如 Dec）

续表

参　数	说　明
%B	月的全称（例如 December）
%c	日期和时间（Mon Nov 8 14:12:46 CST 1999）
%d	一个月的第几天（01～31）
%D	日期（mm/dd/yy）
%m	月（01～12）
%w	一个星期的第几天（0 代表星期天）
%W	一年的第几个星期（00～53，以星期一为第一天）
%x	显示日期的格式（mm/dd/yy）
%y	年的最后两个数字
%Y	年（例如：1970，1996）

6.3.2　规范的 Shell 脚本

虽然懒蜗牛同学用一个类似批处理的脚本文件解决了他每次都要手动敲很多命令的烦恼，但是他写的这个所谓的脚本还是太初级了，一点都不专业。

【要有必要的注释】

一个专业点的脚本，一般总要在脚本里写清楚这个脚本是干什么的。这种给人类看的文字叫做注释，大家应该都听说过吧。

那么我们的 Shell 脚本里怎么写注释呢？很简单，以“#”开头，后面的全是注释。例如懒蜗牛同学的自动备份脚本，就可以加入这样一些注释：

```
#本脚本由懒蜗牛同学设计，用于解脱懒蜗牛同学每周重复备份照片的烦恼
sudo mount /dev/sdb1 /mnt/
cp /mnt/*.JPG /home/lanwoniu/Picture/
tar -czvf /home/lanwoniu/Backup/PIC`date %Y%m%d`.tar.gz /mnt/*.JPG #这一行将自动根据当前日期产生备份文件名，真神奇
rm /mnt/*.JPG
sudo umount /mnt/
```

大致就是这样，只要某一行的某一个位置出现了“#”符号，那么后面的就全是注释了，Shell 不再进行解释。这跟 C 语言中的双斜杠“//”的作用是一样的。

提示：Shell 脚本中没有类似 C 语言中的“/*”，“*/”方式的段落注释。

【指明使用哪个 Shell】

有时候可以看到一些脚本的第一行是类似这样的：

```
#!/bin/bash
```

这又是什么意思呢？有的同学说：我知道，这个以“#”开头，所以只是个注释。但是，很不幸，这不是注释。

确实，我说过，以“#”开头的是注释。但是，“#!”放在一起还出现在脚本第一行，那就不是注释啦！这一行的意思是用来指明这个脚本所需要的 Shell。

前面说过，我们 Linux 系统中有很多的 Shell，比如 bash、tcsh、ksh 等。这些不同的 Shell，他们的特性、语法什么的，大都是不同的。那么一个脚本程序就有必要说明一下，这是个 bash 的脚本，还是 tcsh 或者别的什么 Shell 的脚本。就好像你写了一段代码，总要告诉人家你写的是 C 语言的代码还是 Java 语言的代码吧。

当然，你可以直接告诉系统，说这个脚本是一个 bash 的脚本，请用 bash 来解释这个脚本。那么你就需要这样运行你写的脚本（比如脚本叫做 myscript.sh）：

```
$bash ./myscript.sh
```

但是这显然很麻烦，难道你要自己记住每一个脚本都是对应于哪个 Shell 么（虽然一般情况下都是 bash）？当然不必，就像多数脚本一样，加入对这个脚本所需要的解释器的说明就可以了，比如懒蜗牛同学的自动备份脚本，我们假设它需要 bash 来运行，那就应该完善一下，写成这样（为什么说“假设”需要 bash？因为这个脚本太简单，都是调用命令，基本上是个 Shell 就能正常执行这个脚本）：

```
#!/bin/bash
#本脚本由懒蜗牛同学设计，用于解脱懒蜗牛同学每周重复备份照片的烦恼
sudo mount /dev/sdb1 /mnt/
cp /mnt/*.JPG /home/lanwoniu/Picture/
tar -czvf /home/lanwoniu/Backup/PIC`date %Y%m%d`.tar.gz /mnt/*.JPG #这一行将自动根据当前日期产生备份文件名，真神奇
rm /mnt/*.JPG
sudo umount /mnt/
```

【使用函数】

另外，Shell 脚本也像大多数编程语言一样支持函数。如果你有一段代码需要在脚本里执行多次，不必反复地写多份，而是把它们写成一个函数，直接调用即可。比如这个自动备份的脚本：

```
#!/bin/bash
#本脚本由懒蜗牛同学设计，用于解脱懒蜗牛同学每周重复备份照片的烦恼
backup_picture ()
{
    sudo mount /dev/sdb1 /mnt/
    cp /mnt/*.JPG /home/lanwoniu/Picture/
    tar -czvf /home/lanwoniu/Backup/PIC`date %Y%m%d`.tar.gz /mnt/*.JPG #这一行将自动根据当前日期产生备份文件名，真神奇
    rm /mnt/*.JPG
    sudo umount /mnt/
}
backup_picture #调用函数时不需要写小括号
```

把备份的动作写成一个 backup_picture()函数后，需要进行备份的时候，在脚本里调用这个函数就可以了。脚本在执行的时候，会先略过写进函数里的部分，直到执行到某一行调用了这个函数，再回来执行。

6.3.3 在 Shell 中使用变量

作为一种编程语言，少不了变量。我们 Linux 系统中强大的 Shell 自然也要支持变量。

【用户变量——信手拈来】

Shell 中的变量比较简单、随意。不必声明，随用随写，信手拈来就是一个变量。比如在脚本中有如下语句：

```
value=128
```

这就获得了一个内容是“128”的变量。那么怎么使用这个变量呢？使用变量的时候需要在变量前加上“$”符号，以表示这是一个变量。例如我们写这么个脚本：

```
#!/bin/bash
value=128
echo value
echo $value
```

把这段代码存成一个文本文件（例如 myscript.sh）并赋予可执行权限，然后运行，大约会得到如下的结果：

```
$./myscript.sh
value
128
```

从这个脚本可以看出，要使用一个变量的时候，需要在变量名前加“$”符号以表示它是一个变量，否则就当作一般的字符处理了。

提示： echo 是用于向屏幕打印字符的命令。后面可以直接写字符串，也可以使用变量。

不过有时候我们需要变量紧挨着一个普通字符串，这样可能会产生歧义，例如这个脚本：

```
#!/bin/bash
player1=YOGA
player2=KEN
echo "Game Start! $player1vs$player2"
```

运行之后，并不能获得我们想要的结果，而是打印出了：

```
Game Start! KEN
```

之所以会这样，是因为 Shell 将“$player1vs”看作了一个对 player1vs 的调用。而我们根本没有这个变量（我们只有 player1 变量），所以这里是空白，只打印出了 player2 变量的值“KEN”。如果想让 Shell 理解我们的意图，也很简单，给变量加上大括号：

```
#!/bin/bash
player1=YOGA
player2=KEN
echo "Game Start! ${player1}vs${player2}"
```

这样就可以获得我们想要的结果了：

```
Game Start! YOGAvsKEN
```

【变量类型——只有字符串】

有的同学可能注意到了，我们的脚本里并没有给变量明确声明一个类型。这是因为 Shell 的变量只有一种类型，就是字符串。没有什么整型、浮点型之类的概念。咱们再用一个简单的脚本说明一下：

```
#!/bin/bash
num=8
num=$num + 1
echo $num
```

运行这个脚本，会看到最终打印出来的 num 的值是“8+1”，而不是“9”。因为 bash 这家伙压根就没长数学的脑子！他只会把变量的值作为字符串处理。

可是虽然 bash 没长数学脑子，但是我们的生活不能没有数学啊，遇到需要计算的问题时怎么办呢？没关系，bash 不会算，有人会算，就是 expr 命令。

expr 专门用于 Shell 脚本中，负责对几个字符串变量进行数学计算。比如刚才这个脚本，我们实在是想计算 num+1，看看到底得多少。那么就可以这样：

```
#!/bin/bash
num=8
num=`exprt $num + 1`
echo $num
```

这样就可以如愿看到数字 9 了。

【环境变量——哪都能用】

刚才我们随手定义的变量，可以叫做用户变量。自己定义自己用就好了。除了用户变量之外，还有一个重要的概念，就是环境变量。比如我们之前遇到过的 PATH 变量、HOME 变量等。

所谓环境变量，有点类似于 C 语言里面的全局变量，它在整个系统中都有效。用户变量只在这一个脚本内有效，出了这个脚本，这个变量就没了。而全局变量一旦设定，可以在整个系统中的任何时候、任何地方进行访问。要让一个变量成为全局变量很简单，只要在变量赋值语句前加上 export，类似这样：

```
#!/bin/bash
export env_num=8
echo $env_num
```

运行这个脚本，你自然会看到输出一个“8”，当然这并不是环境变量的特点。环境变量的特点是你运行完这个脚本以后，再输入命令：

```
$echo $env_num
```

依然会看到这个变量的值还是“8”。

提示：环境变量在当前会话结束后失效。

当然，像这样创建出一个环境变量的需求并不多，一般我们在写 Shell 脚本的时候，多数是使用或者修改已经存在的环境变量。比如通过$TZ 变量获取本系统所在的时区；通过$HOME 变量获取当前用户的家目录地址等。比如懒蜗牛同学的自动备份脚本，现在每次都是固定往/home/lanwoniu/backup 目录下备份，很不灵活。这里就可以应用全局变量，改成这样：

```
#!/bin/bash
#本脚本由懒蜗牛同学设计，用于解脱懒蜗牛同学每周重复备份照片的烦恼
backup_picture ()
{
    sudo mount /dev/sdb1 /mnt/
    cp /mnt/*.JPG $HOME/Picture/
```

```
    tar -czvf $HOME/Backup/PIC`date %Y%m%d`.tar.gz /mnt/*.JPG #这一行将自动根据当前日期产生备份文件名，真神奇
    rm /mnt/*.JPG
    sudo umount /mnt/
}
backup_picture #调用函数时不需要写小括号
```

如此一来，就不光是 lanwoniu 用户可以用这个脚本了，任何用户都可以用这个脚本进行备份，提高了灵活性。

【特殊变量——一堆符号】

除了普通的用户变量和环境变量外，还有一些特殊的变量。这些特殊变量特殊在如下几方面。

- 长得就特殊。
- 脚本执行时自动被设定。
- 不可修改。

下面我们就看看这些变量的样子，主要就是下面这些。

- $n——这里，*n* 是一个从 0 到 9 的数字。这个变量代表了执行本脚本所加的第 *n* 个参数。*n*=0 时代表脚本本身的名称。这个变量跟 C 语言中的 argv[]有点类似。
- $*——这个变量代表执行本脚本所加的所有参数（不包括脚本名本身）。
- $#——执行本脚本所加的参数个数，类似 C 语言中的 argc。
- $$——这个脚本的 PID。

【变量赋值】

变量赋值，除了可以直接写出初值之外，还可以将命令的运行结果赋给变量。比如咱们之前用到过的“`”符号，可以调用指令并获得该指令的输出。那么同样也可以把这个输出赋值给一个变量。还拿懒蜗牛同学的备份脚本做例子，可以再这样修改一下：

```
#!/bin/bash
#本脚本由懒蜗牛同学设计，用于解脱懒蜗牛同学每周重复备份照片的烦恼
backup_picture ()
{
    sudo mount /dev/sdb1 /mnt/
    cp /mnt/*.JPG $HOME/Picture/
    tar -czvf $HOME/Backup/PIC${today}.tar.gz /mnt/*.JPG #这一行将自动根据当前日期产生备份文件名，真神奇
    rm /mnt/*.JPG
    sudo umount /mnt/
}
today='date %Y%m%d'
backup_picture #调用函数时不需要写小括号
```

这回我们把当前的日期，存储进了 today 这个变量，这样如果要多次用到日期，就不必每次都调用 date 命令了，直接从变量中读取就可以了。

6.3.4　Shell 中的条件判断

所有编程语言，都少不了条件判断语句。我们的 Shell 也是可以支持简单的条件判断的。

【if 和 fi】

最常见简单的条件判断，就是 if 语句了。Shell 中的 if 语句比较有个性，if 后面的“表

达式”部分必须被足够的空格分隔得分崩离析才可以。比如咱们看下面这个脚本（建议把这个脚本命名为 kill）：

```
#!/bin/bash
if [ "$1" = "me" ]; then
    echo "You are dead…"
else
    echo "$1 is dead..."
fi
```

提示：要注意这个脚本中“[”符号前边和后边、“"$1"”后边、“=”后边、“]”前边，都要有空格。否则脚本错误。

这个脚本的运行效果就是这样：

```
$./kill dog
dog is dead…
$./kill cat
cat is dead…
$./kill me
You are dead…
```

好，我们暂时不去讨论这个脚本中的虐畜及自虐倾向，只讨论里面的 if 语句。if 的工作，就是根据后面命令的返回值，来判断程序应该走哪条分支。另外，if 语句一定要有对应的 fi 作为结尾（相当于 endif）。

这里大家可能有点困惑：if 后面的命令？if 后面不是一个表达式么？哪来的命令呢？

好，那我们写个更简单点的脚本说明一下。

```
#!/bin/bash
if ls -l /home; then
    echo "ls return true! "
else
    echo "ls return false! "
fi
```

运行这个脚本，可以看到 Shell 会调用“ls –l /home”这条命令，列出/home 目录下的文件，然后因为 if 判断到“ls –l /home”这条命令返回了结果“真”（因为命令执行成功嘛），因此显示出了“ls return true!”这个字符串。如果/home 这个目录不存在，ls 命令就会执行失败，返回“假”，于是 if 根据这个返回值判断，程序运行 else 对应的语句，就会打印出“ls return false!”这个字符串。

这下大家是不是明白些了？if 与“;”号之间的，就是一条 Shell 中的命令而已。这个命令返回真，则进入 if 对应的分支；返回假，则进入 else 对应的分支（如果有 else 的话）。那么刚才那个 kill 脚本里的“["$1" = "me"]”这段难道也是命令么？答对了！这就是命令。

“ls -l home”中，“ls”是命令名，后面是两个参数：“-l”和“/home”。

“["$1" = "me"]”中，“[”是命令名！后面有 4 个参数，分别是“"$1"”，“=”，“"me"”，以及“]”。

没错，“[”是一个无比简练的意想不到的命令。这条命令就在/usr/bin/下，用 ls 可以看到。它跟 apt-get、gcc 等命令一样，是一条实实在在的 Shell 命令。这条命令的作用，就是判断后面参数所组成的表达式的值（真或假），并返回。“[”命令要求输入的最后一个参数必须是“]”（这主要是为了你们人类看着顺眼点）。

这样就可以理解一个问题了：既然“[”后面的所谓的变量、常量、“]”号等，对于“[”命令来说都是它的参数，那么这些参数之间必然都要有空格。所以 Shell 脚本中的 if 语句必须写成类似这个样子：

```
if [ $a = $b ]; then
if [ ! $a = $b ]; then
```

而不能写成下边这样：

```
if [$a=$b]; then
if [ !$a = $b];then
```

“[”命令不仅可以判断相等条件，还可以判断很多复杂的条件，如表 6.2 所示。

表 6.2　常用判断参数

文 件 检 查	
[-d $path]	当 path 变量是一个目录且该目录存在时返回真
[-f $file]	当 file 变量是一个文件且该文件存在时返回真
[-e $pathname]	当 pathname 变量是一个文件或目录且该文件或目录存在时返回真
算术比较运算	
[$num1 –eq $num2]	$num1 等于$num2 时返回真
[$num1 –ne $num2]	$num1 不等于$num2 时返回真
[$num1 –lt $num2]	$num1 小于$num2 时返回真
[$num1 –le $num2]	$num1 小于等于$num2 时返回真
[$num1 –gt $num2]	$num1 大于$num2 时返回真
[$num1 –ge $num2]	$num1 大于等于$num2 时返回真
字符串运算	
[$string1 = $string2]	如果 string1 与 string2 内容相同则返回真
[$string1 != $string2]	如果 string1 与 string2 内容不同则返回真
[-z $string]	如果 string 长度为 0 则返回真
[-n $string]	如果 string 长度不为 0 则返回真

提示： “-eq”与“=”的意义不同。“-eq”用于判断数值上的相等，“=”用于判断字符串的完全匹配。例如：$a 的值为“03”，$b 的值为“3”，则“-eq”将判断这 2 者相等，而“=”将判断这 2 者不等。“!=”和“-ne”同理。

有了 if 命令，我们就可以把懒蜗牛同学的自动备份脚本再进一步完善一下：

```
#!/bin/bash
#本脚本由懒蜗牛同学设计，用于解脱懒蜗牛同学每周重复备份照片的烦恼
backup_picture ()
{
    sudo mount /dev/sdb1 /mnt/
    cp /mnt/*.JPG $HOME/Picture/
    tar -czvf $backupfile /mnt/*.JPG #这一行将自动根据当前日期产生备份文件名，
                                     真神奇
    rm /mnt/*.JPG
    sudo umount /mnt/
}
today='date %Y%m%d'
```

```
backupfile="$HOME/Backup/PIC${today}.tar.gz"
if [ -f $backupfile ]; then
    backup picture #调用函数时不需要写小括号
else
    echo "今天已经备份过啦！洗洗睡吧。"
fi
```

【case】

类似于 C 语言的 switch，case 语句，Shell 中也有 case 语句来实现多分支的判断。看看下边这个小脚本：

```
#!/bin/bash
echo "input a number: "
read num
case $num in
one) echo "我认识，这是一";;        #双分号表示本条 case 结束
two) echo "我认识，这是二";;
three) echo "这个我得想想……"       #每个 case 可以有多条语句
     echo "哦，想起来了，这是三";; #一个 case 的最后一条语句一定要用双分号结束。且只
                                    有最后一条语句有双分号
*) echo "这个偶不认得了嘛……";;     #使用*)表示其他值，类似于 default
esac #表示结束
```

这个脚本的运行结果估计您也能看明白，就是个英语没学好的卖萌脚本。通过这个脚本可以比较好地理解 case 的作用：就是用某个变量的值，去匹配下边的几个“)”符号前的字符串。如果某行匹配，则执行该行的语句，直到发现双分号“;;”时停止。如果没有找到匹配的，就执行“*)”一行的内容，遇到双分号时停止。

提示：“*)”一行也可以不存在，则匹配不到任何字符串时就不执行任何命令。

6.3.5 Shell 中的循环语句

【for 循环】

Shell 脚本同样支持 for 循环。不过跟多数语言的 for 循环的写法不太一样的是，Shell 脚本中的 for 循环有种很有个性的格式：

```
for 变量 in 名字列表
do
命令列表
Done
```

其中，“变量”就是一个变量，这个变量一般会在 do 和 done 之间的命令列表中用到。而这个“名字列表”则是一个由空格分隔的字符串列表。Shell 在执行 for 循环时，每次依次从“名字列表”中取出一个字符串赋给“变量”作为变量的值，并执行“命令列表”中的命令。另外，在写 for 语句时，也可以省略 in 及名字列表部分，这表示用当前的位置参数来代替这时的名字列表。

这样说很枯燥，写个小程序吧：

```
#!/bin/bash
for num in 1 2 3 4 5 six
do
```

```
    echo “num = $num”
done
```

运行这个脚本，会看到这样的输出：

```
num = 1
num = 2
num = 3
num = 4
num = 5
num = six
```

这样就很明白了吧。这种方式用于批量处理文件会很方便。当然，如果你需要 C 语言中那种 for 循环，也是可以的。不过写法稍稍有点不一样，要用两个小括号。类似这样：

```
#!/bin/bash
for (( num = 1; num<7; num++))
do
    echo "num = $num"
done
```

这里就不做过多解释了。像其他语言中一样，num 的值从 1 开始，依次累加，直到不满足“num<7”这个条件。

【while 循环】

while 循环也是经常用到的一种结构，它的用法大约如下：

```
while 循环条件;
do
    语句
Done
```

其中，“循环条件”的写法也是和 if 语句一样的。多数情况下使用“[”命令来计算条件并返回结果，这里不再赘述。

无论是 while 循环还是 for 循环，都可以使用 break 和 continue 指令。其中 break 指令用于跳出当前循环体，执行后面的操作；continue 指令用于忽略本次循环，直接回到循环体的开始位置，执行下一次循环。这和其他常用编程语言中的 break 和 continue 是一样的。

6.3.6　扩展阅读：Linux 的文件权限

这一回中，我们提到了写一个脚本，要赋予它可执行权限才能执行。有的同学可能对这个权限还不是很明白，那咱们就仔细说说 Linux 下的文件权限。

【简单的权限——只有 3 种】

熟悉 Windows 系统的同学应该都知道，Windows 下可以对文件设置很详细的权限。谁可以读这个文件，谁可以写这个文件，等等，如图 6.8 所示。

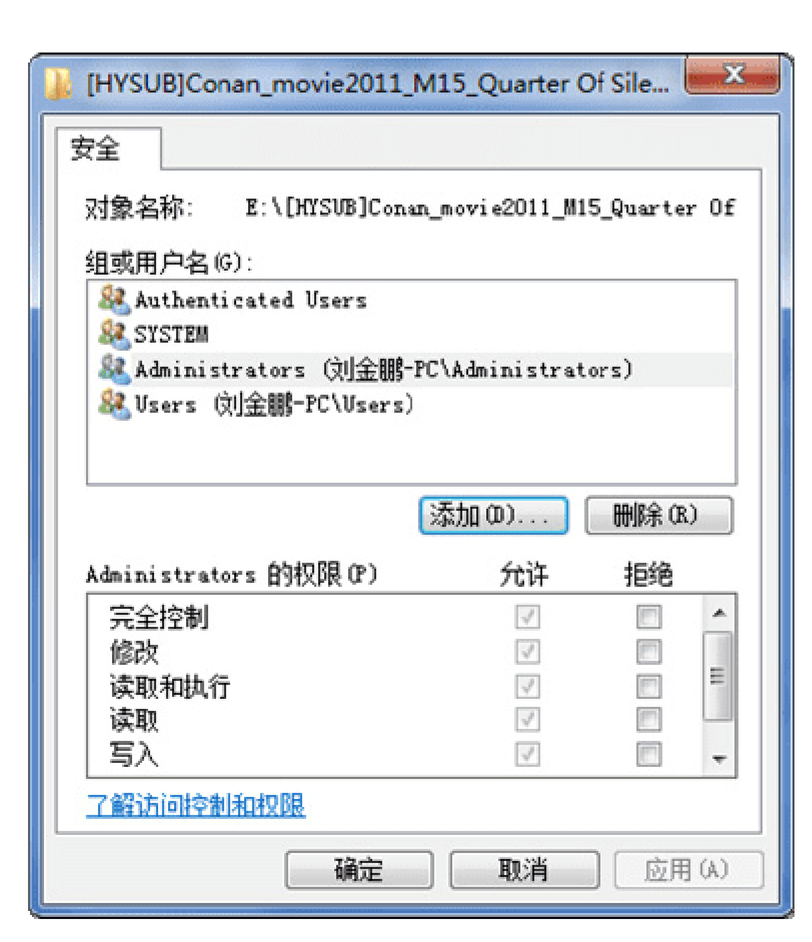


图 6.8　Windows 设置文件权限

我们 Linux 系统中的权限相对简单很多，对于一个文件（包括文件夹），只有 3 种权限——读、写、执行。

对于一个普通的文件，拥有对这个文件的读权限，就是可以读取里面的内容。对于一

个目录（目录也是特殊的文件，在我们 Linux 世界，一切皆是文件），拥有对这个目录的读权限，就意味着可以查看目录中的文件列表（也就是可以用 ls 命令看里面都有什么东西）。

对于一个普通的文件，拥有对这个文件的写权限，就是可以改变里面的内容，增加、修改、删除，这些都算改变。对于一个目录，拥有对这个目录的写权限，就意味着可以删除、移动或者添加目录里的文件或者目录。也就是说，对于目录文件，里面的文件列表就相当于这个目录文件的“内容”，有写权限就是可以修改内容。

提示：要注意的是，对于一个普通文件，有写权限并不代表可以删除这个文件。要对这个文件所在的目录有写权限才可以删除这个文件。

对于一个普通的文件，拥有对这个文件的执行权限，就是可以运行这个文件，比如我们写的脚本。对于一个目录，拥有对这个目录的执行权限，就意味着可以进入这个目录（比如用 cd 命令）。如果一个目录有执行权限，但是没有读权限，就意味着你可以通过 cd 命令进入这个目录，但是进去之后运行 ls 发现什么也看不见。

提示：在赋予一个文件可执行权限时，并不会对这个文件的格式进行检查。也就是说，你完全可以赋予一个 JPEG 格式的图片文件可执行权限而不会收到任何错误提示。当然，当你真的试图运行这个图片文件的时候，肯定会报错的。

【面对的用户——只有 3 类】

现在，这 3 种权限我们明白了。但是权限离不开对用户的识别，Windows 下可以细致地针对某一个用户或者某一个组来分配特定的权限，那么 Linux 呢？本着简洁高效不折腾的原则，我们 Linux 系统只对 3 类用户设置权限。

注意我说的是 3 类，可不是 3 个用户哦。哪 3 类用户呢？就是文件的所有者、群组、其他。

在我们 Linux 系统中，每个文件都明确地属于一个用户。比如我们这里，/home/lanwoniu 目录下的文件基本上都属于 lanwoniu 用户；/etc、/usr、/bin 这些系统目录中的文件，都属于 root 用户等。一个文件所属于的那个用户，我们就叫他所有者吧。Linux 系统中，可以对一个文件的所有者，设置一套权限。比如一个文件叫做“懒蜗牛记账.odt”，属于 lanwoniu 用户。那么可以设定，对于 lanwoniu 用户，这个文件可以被读和写，但不能执行（一个文档而已，当然不能执行）。

文件除了属于一个用户外，还要属于一个组。就好像你家里，你的电脑是属于你的，别人用不了（不知道开机密码）。但同时它也是属于你们家的，你爹虽然不能打开它，但是他有权把它卖掉（让你不好好学习，哼！）。Linux 系统可以对文件所属的组设置一套权限。还比如“懒蜗牛记账.odt”文件，它除了属于 lanwoniu 用户外，还可以属于 family 组。那么就可以设定，对于 family 组的用户，这个文件可以被读，但不能写和执行。

除了所有者和群组之外的用户，就是“其他”了。可以对其他用户设置一套权限。例如除了 lanwoniu 用户和属于 family 组的用户以外，其他用户既不能读，更不能写和执行那个“懒蜗牛记账.odt”文件。就像你家邻居赵大婶完全不能够对你的电脑进行任何处理一样。

【设置权限的命令——chmod】

说了这么半天了，可能有人着急问：到底怎么针对用户设定文件的权限呢？好，下面我们就来介绍这个问题，这需要一个命令——chmod。

chmod 是专门用来修改文件权限的命令，它的使用格式大约是这样：

```
chmod [设置权限的对象]+/-[权限] [文件]
```

其中，“设置权限的对象”，就是指所有者、群组、其他这 3 类。当然你不能在命令里写中文“所有者”，而是用字母表示：u 代表文件的所有者；g 代表文件的群组；o 代表其他。而文件的权限就是读、写、执行，当然也用字母表示：r 表示读，w 表示写，x 表示执行。

比如我想给懒蜗牛日记.odt 文件设置权限，我想给这个文件的所有者（也就是 lanwoniu 这个用户）增加可执行权限（就是个实验，别管这个动作多么抽风），那么就可以运行如下命令：

```
$chmod u+x ./懒蜗牛日记.odt
```

其中，u 代表要对所有者的权限进行操作；+号代表要增加权限；x 代表要增加的是执行权限。那么如果我想去掉 family 组内的成员对这个文件的读权限（日记不能瞎让别人看），那么就可以运行这样的命令：

```
$chmod g-r ./懒蜗牛日记.odt
```

其中，g 代表要对群组的权限进行操作；-号代表要去掉权限；r 代表要去掉的是读取的权限。于是 family 组的成员就不能够读取懒蜗牛同学的日记了。

或者，也可以在图形界面中对文件的权限进行设置。右击要设置权限的文件，在弹出的快捷菜单中选择“属性”，出现文件属性对话框。切换到“权限”标签，就可以看到对文件设置权限的界面了，如图 6.9 所示。

图 6.9　Ubuntu 系统设置文件权限

6.4 正则表达式

在使用 Shell 命令或者 Shell 编程的时候，经常会用到一种叫做“正则表达式”的东西。有了它，很多事情事半功倍。作为一名想成为高手的菜鸟，懒蜗牛同学觉得有必要了解一下这个东西。

6.4.1 什么是正则表达式

正则表达式（Regular Expression）是指一个用来描述或者匹配一系列符合某个句法规则的字符串的单个字符串。听着挺迷茫的吧，那就说简单点：正则表达式就一段火星文似的字符串，这段字符串可以用来表示有一定规律的很多段字符串。

最初的正则表达式，出现在理论计算机科学的自动控制理论和形式化语言理论中（完全听不懂的举手）。在这些领域中有对计算的模型和对形式化语言描述与分类的研究。20 世纪 40 年代，Warren McCulloch 与 Walter Pitts 将神经系统中的神经元描述成小而简单的自动控制元。到 20 世纪 50 年代，数学家 Stephen Cole Kleene 利用称为“正则集合”的数学符号来描述此模型。后来，UNIX 的创始人，Kenneth Lane Thompson 将此符号系统引入编辑器 QED，然后是 UNIX 的编辑器 ed，并最终引入了 grep。自此，正则表达式被广泛地使用于各种 UNIX 或类 UNIX 系统的各种工具中。

6.4.2 初识正则表达式

懒蜗牛同学想学习正则表达式，也不是一时兴起。前几天，他确实遇到了类似的需求。

话说那一天，懒蜗牛拿到一个脚本文件，里面大量使用了 sed 命令。懒蜗牛同学想要复制里面的所有 sed 命令，存入另一个文件里，作为学习 sed 命令用法的参考。经过学习和研究，懒蜗牛了解到有个 grep 命令可以完成这个操作，他的用法大约是这样：

```
$grep <字符串> <文件名>
```

这样的命令所做的事情，就是在“文件名”所指定的文件中，查找带有“字符串”所指定的内容的行，并输出到标准输出。当然，懒蜗牛同学是想存成文件，于是就加了个输出转向，运行了这个命令：

```
$grep "sed" ./learn.sh > ./sed_command.txt
```

提示： “>”可以将原本输出到标准输出的内容（即打印到屏幕上的内容），转向到一个文本文件中。并且如果文件已经存在，它将覆盖掉文件原有内容。如果不想覆盖文件原有内容，可以使用“>>”符号，代替“>”，如此则会在文件末尾追加新的内容。

这样，就把 learn.sh 脚本中，所有带有 sed 字样的行，全部输出到了 sed_command.txt

文件中。但是当懒蜗牛打开 sed_command.txt 查看的时候发现了问题。所有带有 sed 命令的行，固然是都写进来了，但同时还有很多跟 sed 命令无关的行，也跑了进来。比如这一行注释：

```
#I have used this command, but failed.
```

还有：

```
#After my GF kissed me, this line worked!
```

grep 之所以会将这些行匹配出来，是因为它们确实包含了“sed”这个关键字，只不过不像懒蜗牛想象的作为单独的一个命令而已。所以这并不能怪罪 grep 不智能，而是懒蜗牛陈述的要求并不准确。懒蜗牛的要求如果用准确一点的人类语言描述，应该是这样：查找所有包含 sed 作为完整单词的行。

当然你用人类语言说，grep 肯定听不懂。于是，这时候就需要正则表达式出场了。我们需要使用“\b”元字符，这个字符代表了单词的开头或者结尾。那么懒蜗牛要查找的东西，就应该这样表示：

```
$grep "\bsed\b" ./learn.sh > ./sed_command.txt
```

这样，像 kissed、used 这样的词，就不会被匹配到了。这里，我们写的“\bsed\b”就是一个正则表达式。除了“\b”外，还有很多元字符，各代表不同的意义，如表 6.3 所示。

表 6.3　元字符

元　字　符	说　　明
.	匹配任意字符（换行符除外）
\w	匹配字母或数字或下划线或汉字
\s	匹配任意的空白符
\d	匹配数字
\b	匹配单词的开始或结尾
^	匹配行首
$	匹配行尾

提示： 更精确地说，\b 是匹配这样的一种位置——它的前一个字符和后一个字符，有一个是，但不全是\w（一个是，一个不是或不存在）。

6.4.3　强大的正则表达式

又有一回，那位和懒蜗牛同学聊天的 MM 遇到了麻烦。她需要从她写的日记中找到一个固定电话的号码。但是记忆力如此差的她，竟然完全不记得这个电话号码大约出现在哪几天的日记中。于是，她希望能够找出日记中出现过的所有电话号码，然后她根据上下文判断哪个才是需要的。

【使用反义字符】

这样的工作，懒蜗牛同学自然毫不犹豫地包揽下来。在收到 MM 传来的足有 3 MB 大的 diary.txt 文件后，懒蜗牛运行了这个命令：

```
$grep "\d\d\d\d\d\d\d\d" diary.txt
```

咱已经知道了，\d 可以匹配一个数字。这个命令的意思很明白：查找 diary.txt 文件中，所有出现了连续 8 个数字的行。

运行之后，懒蜗牛得到了一些输出，但是好像依旧比较乱。因为一些手机号、QQ 号、各种账号之类的也被搜出来了。它们虽然都是 8 个以上连续的数字，但毕竟它们“包含”了连续的 8 个数字，所以被 grep 找出来了。

那么如果要精确匹配“有且只有 8 位的数字”该怎么办呢？这就要用到正则中的反义了。常用的反义字符如表 6.4 所示。

表 6.4　反义字符

代　　码	说　　明
\W	匹配任意不是字母，数字，下划线，汉字的字符
\S	匹配任意不是空白符的字符
\D	匹配任意不是数字的字符
\B	匹配不是单词开头或结束的位置
[^x]	匹配任意不是 x 的字符
[^xyz]	匹配任意不是 x 且不是 y 且不是 z 的字符

那么，对于懒蜗牛同学的需求，就应该运行这样的命令：

```
$grep "\D\d\d\d\d\d\d\d\d\D" diary.txt
```

这个命令用人类语言描述就是：查找 diary.txt 文件中，所有出现了连续 8 个数字，且此 8 个数字的前后 1 个字符都不是数字的行。这样就能更准确地定位 1 个固定电话的号码了。

【使用重复】

不过这样写连续的 8 个“\d”还是有点累，其实这里可以精简一下，写成这样：

```
$grep "\D\d{8}\D" diary.txt
```

其中，“{8}”的意思，就是前一个字符重复 8 次（\d 看作一个元字符）。类似的重复还有几个，如表 6.5 所示。

表 6.5　常用的重复

代　　码	说　　明
*	重复零次或更多次
+	重复一次或更多次
?	重复零次或一次
{n}	重复 *n* 次
{n,}	重复 *n* 次或更多次
{n,m}	重复 *n* 到 *m* 次

举几个例子吧。比如正则表达式“go*gle”可以匹配“ggle”、“gogle”、“gooogle”、“goooooooooooogle”等。反正就是中间有多少个 o、有没有 o 都没关系；而“go+gle”则

不能匹配“ggle”，它代表必须有至少一个 o；“go?gle”就只能匹配“ggle”和“gogle”；“go{2,4}gle”就只能匹配到“google”、“gooogle”、“goooogle”这 3 种情况了。

【使用中括号】

懒蜗牛终于找出了所有的“出现了连续 8 个数字，且此 8 个数字的前后 1 个字符都不是数字”的行，并让 MM 过目。结果，MM 很不好意思地表示：这些好像都不是，那个电话号码有可能写成了 xxxx xxxx 的格式，也没准是 xxxx-xxxx，或者是带区号的（xxx）xxxx-xxx……

从昏厥中苏醒过来的懒蜗牛同学毫不气馁，继续用正则表达式满足 MM 的需求。区号不区号的先不去管，先看看 xxxx xxxx 和 xxxx-xxxx 怎么匹配吧。其实也简单，使用中括号就可以了，像这样：

```
$grep "\D\d{4}[-\s]\d{4}\D" diary.txt
```

这样的正则表达式用人类语言描述就是“前面有且仅有 4 个数字，中间有一个横杠‘-’或者空白，后面有且仅有 4 个数字的”这么一个字符串。

其中中括号的意思，是表示里面的字符都是或的关系。例如“[abc]”可以匹配 a、b、c 中的任意一个（且仅有一个）字母。或者也可以写一个范围，例如“[a-z]”可以匹配任何一个小写的字母。“[0-9]”就完全相当于“\d”了。

咱们还是举例子吧。比如“b[ae]d”这样一个表达式，就可以匹配 bed 和 bad 这两个字符串；而“[a-c]an”可以匹配 aan、ban、can 这 3 个字符串。

在精确地匹配了 xxxx-xxxx 这种形式的电话号码后，懒蜗牛同学终于找到了 MM 想要的电话。因此，在接下来的几天之内，他一直都对正则表达式赞不绝口。不过，他用到的这点功能，不过是正则表达式中的最初级用法，沧海一粟尔。

6.4.4　无处不在的正则表达式

刚才咱们看着懒蜗牛同学折腾了这么半天，都是在使用 grep 命令时应用正则表达式。其实正则表达式的用途相当广泛，基本上在我们 Linux 系统里，你能想到的地方都能够支持正则表达式。比如常用的查找文件的 find 命令、已经见识过的 grep 命令、编辑字符流的 sed 命令，甚至连 ls 命令都是支持正则表达式的。还有 Vim 编辑中，Emacs 编辑器中，都有支持正则表达式的操作。Shell 编程中更可以用正则表达式了。

可以这么说，在我们 Linux 系统里，只要你觉得某个地方可以用正则表达式来简化操作、提高效率，那么这个地方就一定支持正则表达式。

6.5　多彩的 Shell

懒蜗牛逐渐开始适应了纯终端的操作，于是他有个想法：在这个黑漆漆的界面中，搭建起一个可用的环境，这样以后开机就可以不进图形界面，直接用命令行的软件来做各种事情，可以更高效，更快速。他决定，用一周的时间来完成这件事情。

6.5.1 懒蜗牛同学的计划

起初，懒蜗牛来到混沌漆黑的命令行。

界面是黑漆漆一片，ls 还会出些菱形的方块块，懒蜗牛的手指游弋在键盘上面。

懒蜗牛说：“要有中文！”就有了中文。

懒蜗牛看中文是顺眼的，有中文，有英文，没有了乱码。这是头一日。

懒蜗牛说：“要有声音！打破寂静的黑夜。”

懒蜗牛就让命令行里发出美妙的乐声，这是第 2 日。

懒蜗牛说：“要有窗，看到外面的世界。”于是有了浏览器，懒蜗牛可以重新看到那些网络上的奇花异草，光怪陆离。有了通信工具，懒蜗牛又可以把这些光怪陆离，异草奇花分享给朋友们，这是第 3 日。

懒蜗牛说：“要有色彩，有图，才有真相！”于是就有了图。

图片为黑白的世界带来了色彩。美好的、丑陋的、思念的、怀旧的、唯美的、憧憬的、现实的、抽象的，各种的图片，懒蜗牛微笑了。这是第 4 日。

懒蜗牛说：“要动起来，要鲜活的世界。”于是，就动起来了，这是第 5 日。

第 6 日，懒蜗牛说，我累了，要休息。于是，就没有于是了，这小子没开机。

众人：合着懒蜗牛就是传说中的上帝？

我说：不是，上帝第 7 日才休息。

众人：废话，那是上帝没赶上实行双休日。

我估计这么说您还是听不懂，那我慢慢给您讲解。

6.5.2 命令行下的中文支持

这周一，懒蜗牛开始打造自己的命令行世界。各种命令对懒蜗牛来说都已经很熟悉了，就算有些不大熟练的命令，也可以很轻松地找到纯爷们儿（man）来讲解。但是一来到黑漆漆的终端里面遇到的第一个问题就是，没有中文的支持。

运行 ls，主目录下的图片、文档这些中文目录显示的都是一些菱形的方块。这也难怪，在伪终端下，中文环境由图形部门负责，可到了终端界面中，这事情就没人管了。毕竟多数人还是要使用图形界面的嘛，像我们这位懒蜗牛同学这么神经质的人不多，所以我们没有更多地考虑纯字符界面的使用感受。不过懒蜗牛现在需要字符下的中文环境，那也有人能够胜任，只是默认没有安装而已。而且，这样的软件还不只一个。

【通过 zhcon 显示和输入中文】

首先就是老牌的 zhcon，这个家伙有些岁数了，很久以前他就担当起让命令行下显示和输入中文的重任。

他是一个外挂的中文环境，就像很久以前的 DOS 系统里的那个 UC-DOS 一样。要安装他很简单，找超级牛力就行。

装上之后，在终端里（不是图形界面下的伪终端哦）输入 zhcon，就运行起来了。不过懒蜗牛这么运行了之后发现，这目录下的中文还是有问题啊。原来中文的地方都是菱形

方块，现在倒好，改成一堆问号了。难道 zhcon 的能力就是把方块改成问号？想想也不是啊，于是懒蜗牛运行了 zhcon --help，问问 zhcon 这到底是怎么回事。

zhcon 解释说，你要设置一下编码，我默认是使用 GB2312 的，你系统里的文件都是 UTF-8 的，那你运行的时候就要这样：zhcon --utf8 才行。懒蜗牛这才恍然大悟，心想你倒是早说啊。赶紧照着他说的运行，然后 ls 一下，果然，终于看见中文了。按下 Ctrl+空格组合键，哈哈，出现了输入法，虽然全拼不是那么好用，不过，命令行下，也凑和了。最终的效果如图 6.10 所示。



图 6.10　zhcon 中文环境

【通过 fbterm 显示中文】

zhcon 也是有不少后遗症的，启动了 zhcon 后，凡是一些需要有排版有格式的显示，比如 w3m 显示的网页，moc 显示的 mp3 播放介面，都会或多或少地乱掉。所以懒蜗牛又找了更好的解决方案——fbterm。

fbterm 是一个基于 framebuffer 的终端。framebuffer 是字符界面下程序访问显卡的一种接口。他是能让命令行更加丰富多彩的关键的一环。通过 framebuffer，程序们可以直接向显卡的显存里面更新数据，也就是直接影响屏幕上显示的点和颜色，这就使得在终端下显示图片成为了可能。

提示： zhcon 启动时也可以加入参数“--drv=fb”来指定通过 framebuffer 显示。如果不输入此参数，则 zhcon 自动检测可用的设备。

不过由于程序直接控制显卡的显存，并不经过显卡运算，因此显卡的运算能力是不能发挥的，所有的图形都是靠 CPU 运算好之后直接送到显卡显存里的，所以 framebuffer 比较消耗 CPU。以后我们还会经常看到 framebuffer，现在先说 fbterm。

fbterm 其实也可以算是个虚拟终端，就跟图形界面下的那个 Gnome 终端一样，说不定还是一个学校毕业的呢。只不过 fbterm 是在终端下，用 framebuffer 来“画”出一个虚拟终端来，而 Gnome 终端是靠图形界面组的那些人来画虚拟终端。fbterm 的好处就是不像 zhcon 那样会搞乱屏幕的排版，所以懒蜗牛自从用了 fbterm 之后，就再也没运行过 zhcon。

fbterm 也是源里的软件，直接让超级牛力装就可以。运行简单，就直接运行命令：fbterm 就可以进入能够显示中文的 fbterm 终端了。

提示： 如果 fbterm 需要使用/dev/fb0 设备（framebuffer 设备），当前用户对该设备没有访问权限，需要把用户加入 video 组。运行命令：

```
$sudo gpasswd -a <用户名> video
```

即可，或者使用图形界面操作也可以。

不过 fbterm 只是一个能够显示中文的终端而已，不是一个中文环境。所以，只能看到中文，输入是不行的。不过没关系，专门在命令行下的输入法也有，和 fbterm 配合起来就天衣无缝了。

【yong 输入法】

关于中文输入法，推荐来自中国的 yong，学名叫做“小小输入法”，这个我们的源里可没有，得去他的网站上下载并安装，就是这个地址：http://yong.uueasy.com/read.php?tid=1。

装好之后，只要在运行 fbterm 的时候加上参数，指定输入法，就可以在 fbterm 中使用了，指定输入法用-i 参数：

```
$fbterm -i yong
```

就可以了。

除了 yong，还有 fbterm_ucimf 可以使用，不过相对简陋些。另外还可以使用 ibus-fbterm，看名字就知道了，是基于 ibus 的，相对好用些。

最终懒蜗牛选择了 fbterm，这不单是因为看上去顺眼点，更关键的是后来的境遇。

【开启 framebuffer】

咱说了半天 framebuffer，可能有同学要问了，这个东西怎么开启呢？别忙，我给您讲讲。

我们 Ubuntu 的终端下默认并没有开启 framebuffer（还是那句话，毕竟用终端的人少），如果需要开启，跟着我执行以下操作：

（1）打开/etc/initramfs-tools/modules 文件，加入以下两行：

```
fbcon
vesafb
```

这是开启 framebuffer 所需要加载的驱动。

（2）打开/etc/modprobe.d/blacklist-framebuffer.conf 文件，删除或注释掉带有“vesafb”字样的行。

（3）执行命令：

```
$sudo update-initramfs -u
```

此命令将更新 initramwfs 文件。

（4）打开/etc/default/grub 文件，找到如下一行：

```
GRUB_CMDLINE_LINUX_DEFAULT = "quiet splash"
```

将其改为：

```
GRUB_CMDLINE_LINUX_DEFAULT = "quiet splash vga=789"
```

这一行是设置内核启动时的参数，那个 vga=786 是告诉内核（也就是告诉我啦）启动

的时候要使用什么样的分辨率和色深。具体这个数字应该写多少，可以参见表 6.6。这里的 789，就是 800×600 的分辨率，24 位色深。

（5）更新 grub，运行命令：

```
$sudo update-grub
```

（6）重新启动系统就可以了。

表 6.6　framebuffer显示模式

	640×480	800×600	1024×768	1280×1024
8 bit	769	771	773	775
15 bit	784	787	790	793
16 bit	785	788	791	794
24 bit	786	789	792	795

6.5.3　在 Shell 下播放音乐

周二，懒蜗牛得到了比较顺手的有中文环境的命令行之后，又开始踏上了新的寻觅旅程——寻觅音乐。

在图形界面下的时候，有很多播放软件可以让懒蜗牛浸泡在音乐的海洋里。那在字符界面下呢？想想也不应该有问题呀，放音乐又不需要图形界面是不是？字符界面下播放音乐还是不成问题的，Mplayer 就可以播放各种音频，还有个 Mpg123 也可以放出声来。不过他们俩虽然能放出声音来，毕竟不专业，怎么也得有个播放列表、歌曲管理的功能吧。于是，懒蜗牛选择了 moc。

提示： Gnome 界面下，鼠标悬停在音乐文件上就可以预览其内容，这个功能就是通过调用 Mpg123 实现的。

【终端里的窗口】

moc 是一个字符界面的音乐播放软件。有好奇而 OUT 的同学可能会猜想：字符界面播放音乐，那是不是要输入命令才播放音乐呢？比如输入 play，就播放；输入 stop 就暂停；输入 load xxxx.list 就导入播放列表这样？要是 30 年前没准能有这种软件，不过在现在这么简洁高效的年代，怎么可能用这么难用的东西呢？虽说是字符界面，但是不代表就不能有窗口！是的，你没听错，在字符界面下也能画出窗口来！怎么画？用字符拼！

我们知道，ASCII 码中有一些特殊的字符，什么横杠、竖杠、拐弯杠什么的。用这些特殊的符号，加上可控制的字符底色，可以拼接出有窗口效果的终端显示的界面。

当然，如果每个程序都自己写代码去拼肯定是很费事的，所以崇尚共享的我们 Linux 系统提供了一套专门在字符界面下画窗口的函数库，叫做 ncurses。调用这个库画窗口就跟用 gtk+提供的接口在图形界面下画窗口类似，所以编程人员不必在意怎么用各种字符拼出好看的窗口，只要调用 ncurses 就行了。ncurses 的前身是 curses，大名鼎鼎的 Vi 就是用它实现的界面。

【moc 的操作】

moc 就是一个基于 ncurses 的、字符界面的音乐播放软件。直接找超级牛力就可以安装。

懒蜗牛装了之后赶紧运行一下试试。虽然包名叫 moc，不过运行的命令是 mocp。运行起来之后如图 6.11 所示。

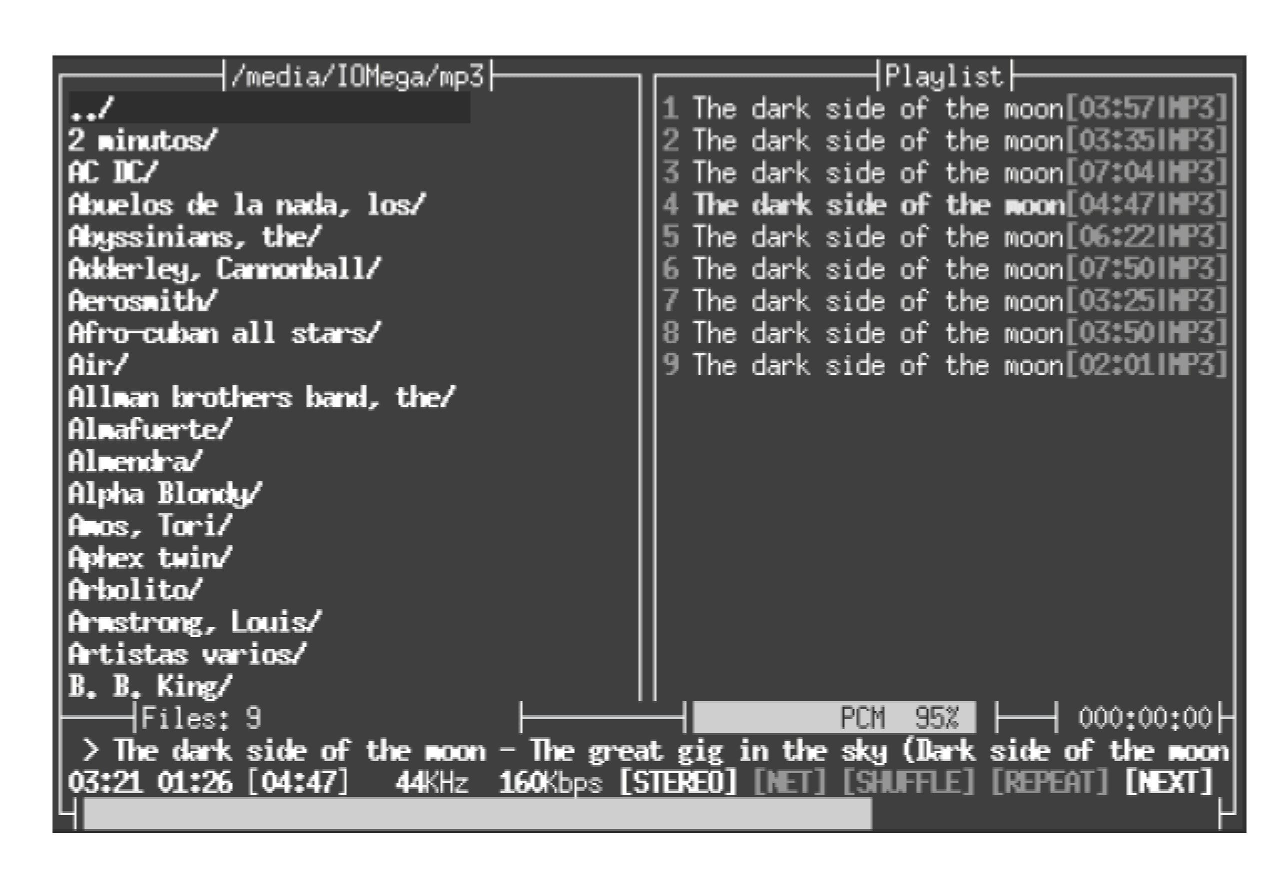


图 6.11　mocp 运行界面

运行起来之后，左边是文件列表窗口，可以通过上下箭头选择相应目录，按回车键进入，最上面的那个“..”表示上一级目录，这个都知道吧，不知道的面壁去。

选到要听的 MP3 文件按 A 键将文件加入右边的播放列表。把 MP3 都加进来之后，用 Tab 键，切换左边的文件列表和右边的播放列表。按回车键就可以播放了。然后还有些快捷键比较常用，介绍如下。

- h 键，打印出使用说明。
- 左右方向键可以调节播放的进度。
- “,”、“.”用来控制音量，“,”键为减小，“.”键为增大。
- s 键停止播放。
- b、n 键分别是跳到“上一首”、“下一首”曲目。
- R 和 X 键用来控制重复播放和循环播放（注意大小写的区别，所谓按 R 键，就是按下 Shift 然后按下 r 键，也就是在终端打出大写的 R 的操作顺序）。

有人说，这个播放软件把整个屏幕都占了，我还怎么干别的呀？简单，你有如下 3 个选择。

（1）不干别的了。

（2）按下 Ctrl+Alt+F2 组合键，换个终端。

当然，这两个方法都是比较弱智的，呵呵。

（3）其实 mocp 是一个 Server 和前端分离的软件，你可以按 q 键退出前端界面，然后该干啥干啥，音乐继续播放，想再回来就再运行 mocp。如果想彻底退出，就按 Q 键，也就是按住 Shift 再按 q 键。

好了，懒蜗牛已经添加好播放列表，按下回车键，音箱里传来了悠扬的乐曲。

6.5.4　在命令行中上网

周三这天，懒蜗牛要解决一个关键的应用——上网。平时在图形界面下，最忙碌的就是狐狸妹妹 Firefox，基本上只要电脑开着，内存里就少不了狐狸妹妹的身影。要是命令行

下不能上网，那还不得把懒蜗牛同学憋屈死。

不过命令行毕竟是命令行，功能还是有限的，所以对浏览器的要求也不能太高，简单的文字的网页还是没问题的。图片呢，实时的显示也不行，不过勉强可以调用别的程序来看一下。但是像 Flash、在线的视频音频这类的就不用想了。好在懒蜗牛同学比较通情达理，也就是想看看文字的东西和一些简单的图片，所以，有那么几个候选软件是可以满足懒蜗牛要求的：w3m, lynx, links。

【命令行中的浏览器——w3m】

这 3 个软件都是可以直接叫超级牛力去装的，很省事。但经过懒蜗牛的试用，lynx 和 links 对中文的支持都相对差点，并且他们都不能显示图片，所以被 pass 了，w3m 华丽胜出。不过 w3m 也不是天生就会显示图片，他需要一些工具，这些工具被打成一个包放在软件源里，包的名字叫做 w3m-img，懒蜗牛让超级牛力给 w3m 搬来了这个工具包之后，w3m 才拥有了显示网页上的图片的能力。

当然，w3m 显示图片跟 moc 显示窗口可不一样，可不是拿特殊字符拼出图片，如果是那样，整个屏幕顶多拼个超级玛丽出来，连蘑菇都没地方显示了，估计就是图 6.12 所示的这个效果。

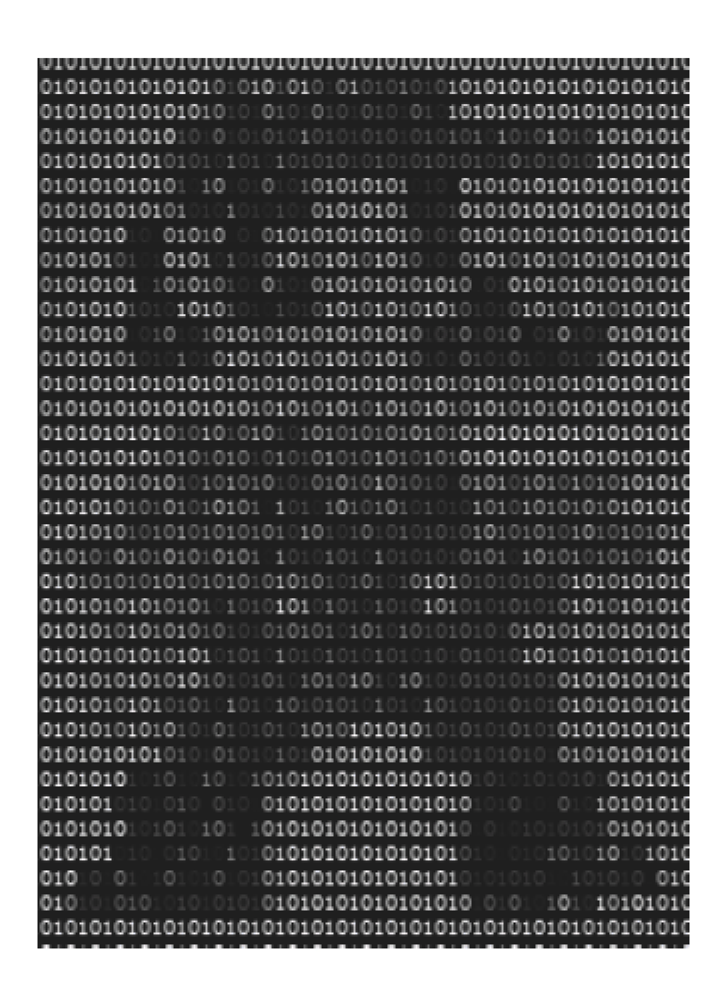


图 6.12　字符拼成的超级玛丽

要显示正经的 JPG 这样的图片，还是需要 framebuffer 的支持。framebuffer 基本上是能够使命令行界面变得多姿多彩的最基本、最重要的条件。懒蜗牛要访问的网页无疑基本上都是中文的，所以需要能够支持中文的终端。咱之前说过了，就是 fbterm。于是，懒蜗牛就在 fbterm 下运行起了 w3m。

【jfbterm 中的 w3m】

不过，天有不测风云日，人有无意失手时，软件也有偶尔不大正常的时候。

虽然在某些系统某些机器上，fbterm 下的 w3m（前提是装了 w3m-img 哦）确实可以正常地显示出图片，不过不知道为什么在我们这里，这个 w3m 和 fbterm 配合了半天也没配合上。他们一运行起来就听着他们俩不停地吵吵：

“fbterm，快点把 fb0 设备给我，我要显示网页上这个图片。”

“你怎么又要啊，我这边的中文还没显示好呢。”

“你怎么那么多中文要显示啊？”

“你这不废话么，我显示的都是你的网页，显示多少中文还不是你说了算。”

“别急别急，你显示完了赶紧给我用。”

“等等，等等，再有 3 毫秒就完了。”

“靠，懒蜗牛翻篇了。刚才那个图片翻过去了，白算了半天。”

“好了我用完了，给你用去吧。”

“用什么用，他已经翻到下一页了，又一个新的图片，我先得解码看看这个图片里是什么才知道该显示什么嘛。”

总之，这俩软件从来没对上过，于是懒蜗牛就没看到图片。

提示： Ubuntu 10.04 中，开启 fbterm 后会因/dev/fb0 设备被占用而导致 w3m 无法显示图片。

不过后来懒蜗牛还是找到了解决办法，那就是换了另一个终端，其实跟 fbterm 还有点关系，叫做 jfbterm，也一样可以支持中文。在 jfbterm 下，w3m 终于可以正常显示图片了，虽然效果肯定不如狐狸的好看，但能在不开图形界面的情况下看到图片，也算奢侈了，就是类似图 6.13 这样的效果。



图 6.13　用 w3m 访问 Google

6.5.5　在 Shell 下看图片

周四，懒蜗牛又一次摄影归来，插上 SD 卡，运行 mount 命令挂载并且把照片 cp 到主目录中的照片文件夹。之后，就开始研究命令行下到底用什么来看照片了。

要看照片无疑还是需要 framebuffer 的支持，这一点咱就不说了。那么那个能够利用 framebuffer 设备来显示图片的软件是谁呢？那可是一个大名鼎鼎的家伙——fbi！

别激动，这个 fbi 不是某大叔的啥啥调查局，只是 Linux 下的一个 FrameBuffer Image Viewer，也就是基于 framebuffer 的图像查看器而已。这个查看器可以查看各种常见的图片格式，全屏显示、缩放、幻灯片模式播放、旋转，都没问题，更复杂的什么调节对比度、亮度啥的自然就不行了，毕竟人家只是个查看器嘛。

懒蜗牛复制好了照片后，用 cd 命令切换进入存着照片的目录，运行了：

```
fbi ./*.jpg
```

这个命令大家应该能看懂吧，就是让 fbi 去查看当前目录下的所有的 JPG 文件的意思。顺便复习一下，“*”这个通配符是由 bash 来处理的，就是说，懒蜗牛输入了这个命令后，bash 先把命令改成：

```
fbi ./123.jpg 456.j GSMLY.jpg WTL.jpg CJK.jpg BDYJY.jpg FDA.jpg
```

这样的形式，然后再按照这个命令去叫醒 fbi，告诉他要去显示这么多图片。然后就是 fbi 起床，一个一个地显示这些图片，当懒蜗牛按“-”、“+”键的时候就对图片进行缩小、放大。按下 PageDown 键就播放下一张，偶尔遇到竖着拍的照片可能还需要按 r 或者 l 键旋转一下，总之，懒蜗牛在悠然轻松地欣赏照片的过程中度过了一晚上。

6.5.6　在 Shell 下播放视频

光看静态的照片还是有些不过瘾的，于是周五，懒蜗牛同学复制来了一个叫“lanjingling.rmvb”的文件，看名字好像是叫什么“懒精灵”？真是什么人看什么片。

【观看本地文件】

那命令行下能看片么？答案依然是肯定的。当然，也依然离不开 framebuffer。平时在图形界面下懒蜗牛经常用来看片的播放器是 SMplayer，当初咱们介绍这个家伙的时候就说过，他只是个前端界面，真正在后面默默无闻地进行播放工作的是 Mpayer，Smplayer 能播放什么文件完全取决于 Mplayer 能播放什么文件，只是因为 Mplayer 总是在后台，人们都忽略了他的存在。现在，到了字符界面，Mplayer 终于可以从后台来到前台了。懒蜗牛用 cd 命令来到了存放 lanjingling.rmvb 这个文件的目录里，运行了：

```
$mplayer ./lanjingling.rmvb
```

于是，一个动态的画面出现在了黑漆漆的屏幕上，一群蓝皮肤白帽子的小家伙映入了懒蜗牛同学的眼帘。我猜懒蜗牛同学一定最喜欢里面那个叫惰惰的家伙。

提示：如果不添加任何参数运行“mplayer 路径/视频文件”，则只会按照视频的原始大小显示，且此时按 f 快捷键无法全屏。效果类似图 6.14 所示。
如果需要全屏播放，需要添加“-fs”参数，例如：

```
$mplayer -fs ./lanjingling.rmvb
```

【观看流媒体】

除了硬盘里现成的视频文件以外，Mplayer 还可以播放 mms 流媒体，也就是一些在线的视频也是可以播放的。

懒蜗牛找到了一些网上的电视台的流媒体地址，什么 CCTV5 啊，CCTV8 啊，这 TV 那 TV 的，反正我是不知道什么内容，但是懒蜗牛看得很起劲。流媒体的地址大约就是 mms://URL/FILENAME 这样的形式，要看这样的媒体可以简单地运行：

```
mplayer mms://URL/FILENAME
```

好像懒蜗牛同学屋里也没有那个叫做 TV 的东西，于是有了 mplayer，他就拿电脑当作 TV 用了。



图 6.14　在命令行下播放视频

6.5.7　扩展阅读：bash 的发展历史

【Thompson Shell】

在最初创造 UNIX 系统的时候，Ken Thompson 大牛在 Bell 实验室写了一个简单的程序，将它作为新的 UNIX 操作系统的接口界面来让系统和人类进行交流。这个程序就叫做 Shell，那时候还没正式起名字，为了让使用这个 Shell 的人们不忘记作者的辛劳，大家就管它叫 Thompson Shell 了。

Thompson Shell 的功能很简单，用户通过它输入一些指定的命令，它负责解释为需要计算机做的操作，并去执行。另外它还能够支持一些简单的脚本，就是把一堆命令写进一个文件里依次执行。但并没有更高级的例如流程控制、分支、变量、函数之类的东西。

【Bourne Shell】

同时，Ken Thompson 的同事，同在 Bell 实验室的 Steve Bourne（就是图 6.15 所示的这位）也写了一个 Shell 程序。作为一个有很多重要程序要写的大牛，Steve Bourne 也没来得及给他写的 Shell 起名字，于是人们同样为了不忘记作者的贡献，管这个 Shell 叫做 Bourne Shell。这个 Shell 跟 Thompson Shell 不同，他加入了流控制，可以写简单的函数。

【标准 Shell 之争】

等到了 20 世纪 70 年代晚期，每种 Shell 在 Bell 实验室都有不少人用，于是形成了两个派别。说来这两个 Shell 应该都是很不错的，用熟悉了都很顺手，但是这两个 Shell 是互不兼容的，就如同修习了少林的内功后再去学武当的心法，多半不是很习惯。

但是作为一个成熟的商业化的系统，总该有个默认的、标准的 Shell 才方便用户学习使用。于是在 Bell 实验室里，分别支持 Thompson Shell 和 Bourne Shell 的两大帮派，进行

了激烈的辩论，经过 3 次连续的 UNIX 用户组集会上两大帮派的斗争之后，终于确立了 Bourne Shell 成为 UNIX 的标准 Shell。



图 6.15　Steve Bourne

【Bash 诞生】

等到 1978 年，Bourne Shell 随着 Version7 UNIX 一同发布，终于告别了实验室，和广大用户见面了。9 年后，1987 年，一个叫做 Brian Fox 的家伙非常喜欢 Bourne Shell，并且觉得它还可以更加完善，于是开始在 Bourne Shell 的基础上进行创造，几年后它成为一个更加完整而且好用的 Shell。出于对 Bourne Shell 的缅怀和崇拜，他将这个 Shell 命名为 Bourne Again Shell——简称 bash。现在，bash 是绝大多数 Linux 系统及 Mac OS X v10.4 系统的默认 Shell。甚至还被移植到了 Windows 系统上，什么？你没见过？那你听说过 Cygwin 吧，那里面就是 bash。

6.6 本章小结

这个懒蜗牛同学对于 Linux 系统，可说是越用越有感觉了。在图形界面下玩腻了就跑到字符界面下感受了一番。他记住了一些简单的命令，还学会了编写简单的 Shell 脚本和使用正则表达式。但这不是主要的，更重要的是了解了命令行的很多基本的概念，也可以感受到 Linux 系统的一些优秀的设计思想。有了命令行的使用基础以后，懒蜗牛接下来会去学点什么呢？咱们下回再说。

第 7 章　改造这个世界

懒蜗牛同学熟悉了命令行的操作，了解了各种命令之后，越来越能体会到我们 Ubuntu 系统的自由和开放。但他没有就此结束探索，他还要更深入地学习，学习编程，学习如何改造这个 Linux 世界。

7.1　C/C++语言开发环境的搭建

编程语言有很多种，懒蜗牛也不知道哪个适合自己。所幸他上学的时候学过 Turbo C，而且听说 C 语言在 Linux 里用得很广泛，那就从 C 语言开始吧。

7.1.1　安装开发套件

C 语言是当年创造 UNIX 系统的主要参与者之一——Dennis M. Ritchie 设计并实现的。（就是图 7.1 里所示的这位）当年这位牛人在写 UNIX 系统的时候，觉得没啥顺手的编程语言，于是就基于 B 语言进行改造，设计出了 C 语言。

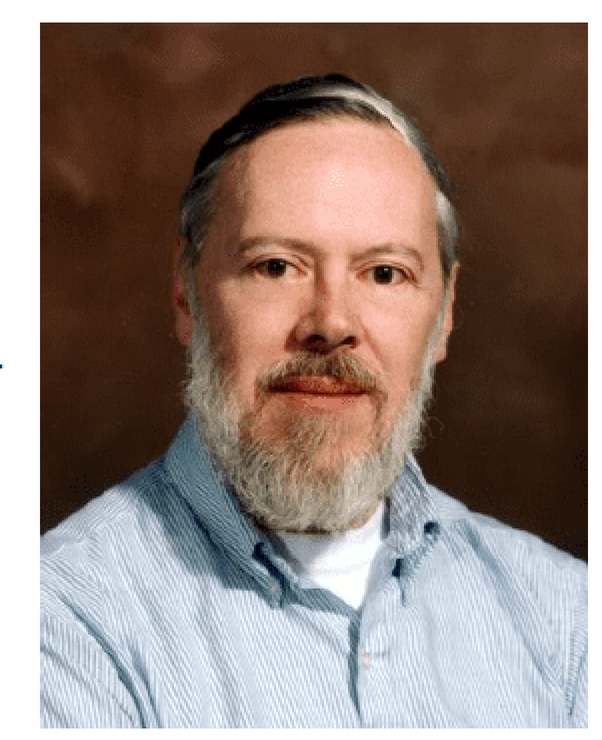


图 7.1　Dennis M. Ritchie

牛人设计出的东西总是很经典的，C 语言就是一个经典中的经典，经典到至今仍然被广泛应用于各种编程的场合。C 语言是一种编译型语言，所以要使用 C 语言开发程序，就需要安装编译器。

提示： B 语言是由 BCPL 语言发展而来的，设计者是美国贝尔实验室的 Ken Thompson。将它取名为 B 语言的意思是将 BCPL 语言进行压缩，提炼出它的精华。

【编译型语言】

有人问，什么叫编译型语言呢？

编程语言大致分成两种，编译型和解释型。编译型语言，就是像 C 语言这样的，写完了需要用编译器编译的语言。

C 语言的程序源码，对我们系统来说就是一个普通的写满字符的文本文件而已。就比如你看书懒得自己翻页（这得多么懒啊），于是要设计一个自动翻页的机器，那么肯定要先有机器的大致的设计图纸。C 语言的源代码就相当于是设计图纸，而最终要得到的那个可以运行起来的程序就相当于最终的翻页机。要想让图纸变成机器，需要一个制造的过程，这个过程就相当于编译过程。造机器的过程中肯定需要各种工具，什么扳手、钳子、螺丝刀之类的。软件的编译也需要工具，就是编译工具。

通过编译工具的编译，原本一个或者几个源码文件（多数情况下是很多源码文件），就变成了活生生的、欢蹦乱跳的、能够像狐狸妹妹（以下“狐狸妹妹”或“狐狸”特指 Firefox）一样跑进内存运行的程序。

除了编译型语言外，还有一种解释型语言，咱们以后再解说它。这里先说 C 语言的编译工具。

【GCC 的来历】

我们 Linux 系统中最常用的 C 语言编译器就是 GCC 了，图 7.2 所示是它的 Logo。

图 7.2　GCC 的 Logo

GCC 这个项目是 GNU 计划的发起者——Richard Stallman 于 1984 年开始实施的。到 1987 年，首次发行了最初的 GCC 版本。他最初取名 GCC 是想表示 GNU C Compiler，也就是 GNU 系统中的 C 语言编译器。GCC 起初是在早期的 Sun 和 DEC VAX 系统上运行的，由于它是开源的编译器（Richard Stallman 创造的东西，怎么可能不开源，除非太阳从西边出来），因此热心的爱好者们可以随意修改并且完善它。这其中最主要的工作，就是提供对各种处理器架构 Arm、MIPS、x86 等的支持。很快，GCC 就能够支持大多数流行的（甚至罕见的）处理器架构了。表 7.1 列出了目前 GCC 支持的 CPU 架构。

表 7.1　GCC支持的处理器结构

较知名的处理器架构（以 4.1 版为准）	
Alpha	ARM
Atmel AVR	Blackfin
H8/300	IA-32（x86）与 x86-64
IA-64 例如：Itanium	MorphoSys 家族
Motorola 68000	Motorola 88000
MIPS 与龙芯	PA-RISC
PDP-11	PowerPC
System/370，System/390	SuperH
HC12	SPARC
VAX	Renesas R8C / M16C / M32C 家族
较不知名的处理器架构	
A29K	ARC
C4x	CRIS
D30V	DSP16xx
FR-30	FR-V
Intel i960	IP2000
M32R	68HC11
MCORE	MMIX
MN10200	MN10300
NS32K	ROMP
Stormy16	V850
Xtensa	

由此可见，GCC 支持的处理器还是非常广泛的。不仅如此，随着开源贡献者的不断完善，GCC 的功能还得到充分的扩充，不仅可以编译 C 语言，什么 Ada、Fortran、C++、Object-C，它都可以支持。它不再仅仅是一个 C Compiler，于是它的首字母缩写的意思就变成了 GNU Compiler Collection，也就是 GNU 系统编译器套装。

🔔**提示：**Object-C 是扩充 C 的面向对象编程语言。它主要使用于 Mac OS X 和 GNUstep 这两个使用 OpenStep 标准的系统。

除了 GCC 外，Intel 公司还开发了一个专门针对他们公司的处理器进行优化的编译器——ICC，大约就是 Intel C Compiler 的意思吧。如果你用的是 Intel 公司的 CPU，那么这个编译器编译出来的东西，理论上效率会高一些。不过这个编译器目前还不大靠谱，用来试验还可以。

【需要安装的软件包】

好，咱还回来说 GCC。说了这么多，这个 GCC 编译器怎么安装呢？是不是叫超级牛力来装就好了呢？告诉您，超级牛力都不用，装系统的时候就装上了。那么既然已经有了编译器，懒蜗牛同学是不是可以马上开始用 GCC 编译 C 语言的程序了呢？也不是，只有 GCC 是不够的，还需要一些必要的头文件和库文件。

前面说过，GCC 就相当于一个用来加工代码的工具，就像木匠造凳子需要的凿子、斧子、锯子这些工具一样。但光有工具不行，还得有材料。要创造东西总是要把某种东西经过加工才变成成品的，总不可能凭空创造出东西吧。你看木匠要造凳子得需要木头、钉子或者胶水这些材料。根据爱因斯坦的物质守恒学说……哦，有点扯远了，总之，要创造程序，需要工具和材料。

用户要创造一个程序，他需要的工具，就是 GCC。而需要的材料，就是各种头文件、库文件这样的文件。创造程序之前，需要准备好这些东西才可以开始。我们的懒蜗牛同学似乎已经进行了充分的学习，对这些理解得比较透彻，所以他就直接叫超级牛力来帮忙准备好这些工具和材料。

为了方便用户安装开发环境，超级牛力已经把创造程序需要的工具都打好了包，包名就叫做 build-essential。所以懒蜗牛运行了：

```
$sudo apt-get install build-essential
```

就去把这些东西从网络上拖回来并且安装好了。作为仅仅是初学 C 语言的懒蜗牛同学来说，装好这个包就够了。

🔔**提示：**如果编写较为复杂的程序，则需要根据程序所涉及的功能安装相应的库。例如，编写需要调用 OpenGL 绘图的程序，则需安装 OpenGL 库；编写 gtk 图形界面程序，则需安装 gtk 库等。

7.1.2　在哪编写程序

装好了编译工具，懒蜗牛还是有些不知从何下手。他在图形界面找了半天也没找到 GCC 到底装哪里了。嗯！看来是个命令行程序，于是懒蜗牛又在命令行下运行 GCC，结果提示说：no input files，这到底是什么意思呢？连个界面都没有，往哪里写程序啊？

【散装的工具和成套的套件】

看来懒蜗牛同学已经习惯了 Windows 下的开发工具了，以为 GCC 跟 Turbo C 或者 Visual C++一样呢。其实人家 Windows 下的 Visual C++是一个集成开发环境，那可是个巨无霸型的软件。他拥有编译程序、编辑文本、项目管理、程序调试、帮助文档等各种各样的功能。这符合微软公司总是把软件做得大而全的一贯作风。装了这么一个软件，您就用去吧，啥都有了。

而在我们崇尚简洁的 Linux 系统中，这些功能分别由不同的软件去实现。例如帮助文档由 man 来负责；项目管理靠用户写 Makefile 来实现；文本的编辑则由任何一个用户用着顺手的文本编辑器完成；编译程序，才是 GCC 的工作。还是那句话：只做一件事，但要做到最好！

所以，写程序这件事其实可以用任何一种文本编辑器完成，只要是能编写文本文件的，都可以写 C 语言的程序。

【gedit】

您别听着“文本编辑器”这个名词觉得很高深，其实您早就认识并且使用过了，像我们这里的 gedit 小弟，或者 Windows 7 那里的记事本，都是文本编辑器。别看 gedit 小弟个头不大，论本事可比 Windows 7 那个记事本厉害多了。至少人家能够认识一些基本语言格式，什么 C 语言，脚本语言之类的。用 gedit 打开一个 C 源码文件，他会将程序中的一些关键词、常量、变量之类的用不同的颜色显示出来并加以区分，这样看起来就比较清楚，如图 7.3 所示，是挺好看的。

【kate】

还有一个跟 gedit 类似的文本编辑器，就是 KDE 下默认的文本编辑器 kate。kate 是 KDE Advanced Text Editor，也就是 KDE 高级文本编辑器的缩写。这个家伙支持 C、C++、HTML 等语言的语法高亮，还支持函数折叠，比 gedit 先进一些。样子看着也专业一点，如图 7.4 所示。



图 7.3　gedit 显示 C 语言代码

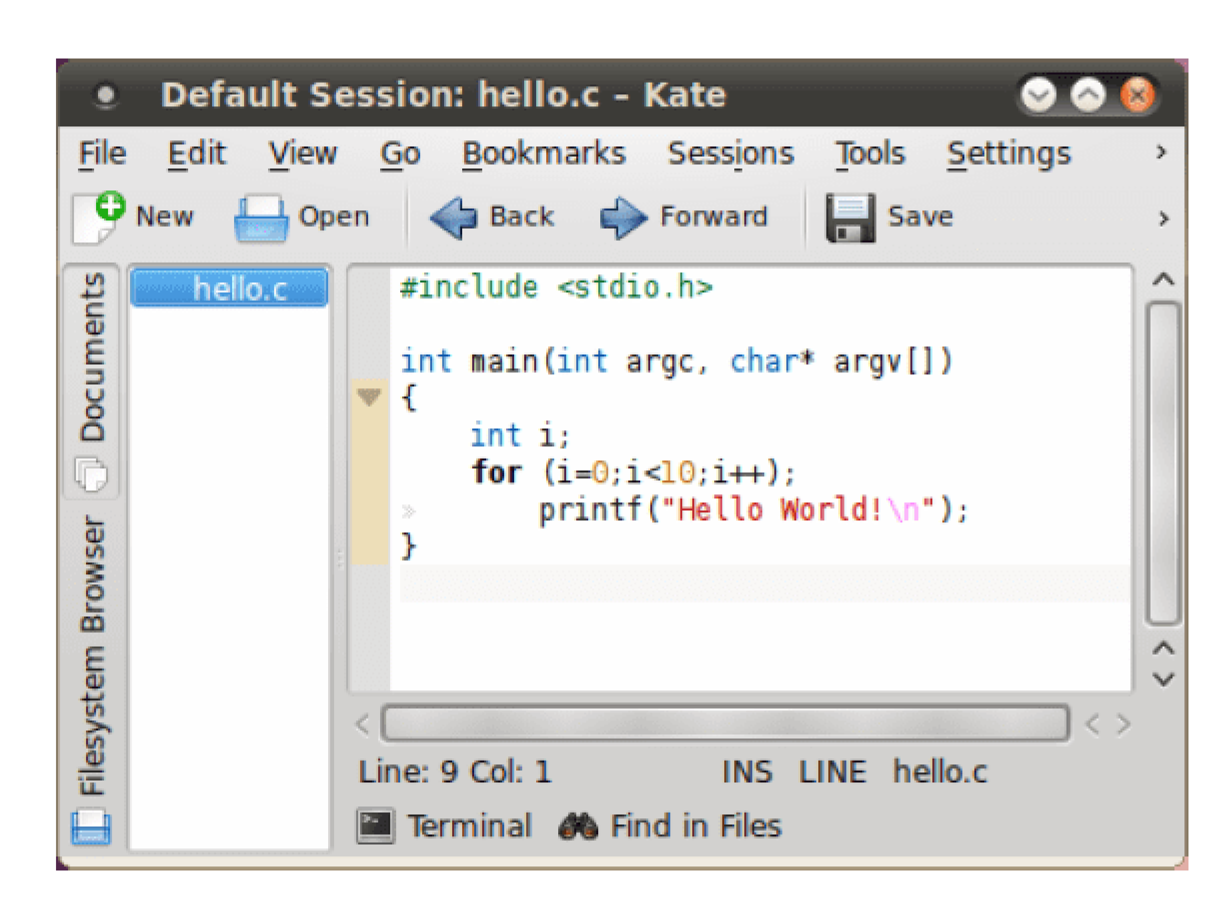


图 7.4　kate 显示 C 语言代码

提示：KDE 下的集成开发环境 KDevelop 就是调用 kate 来实现代码的编写。

【Vim】

上面说的编辑器虽然能够写点小程序，有点小功能，不过毕竟有限。真要做开发，还

是有些应付不了。这时候就需要更强大的文本编辑器了，Vim 就是一个。

Vim，即 Vi Improved，它是 Bram Moolenaar 开发的、与无比强大且无比难用的 Vi 编辑器相兼容并且是更加强大易用的文本编辑器。它支持语法变色、正规表达式匹配与替换、插入补全、自定义键等功能，为编写程序提供了极大的方便。图 7.5 所示是它的 Logo。

我们 Ubuntu 系统中默认带有 Vi 编辑器，可能很多同学已经听说过它的大名了。不过，这个编辑器实在是很不好用，太没人性了，所以一般人都会让超级牛力去安装一个更人性化一点的 Vi——这就是 Vim，可以这样安装：

```
$sudo apt-get install vim-gnome
```

安装这个包之后，就有了字符界面的 Vim 和图形界面的 gVim 可以使用。使用方法基本一样，所不同的就是 gVim 支持一些鼠标操作。如图 7.6 所示就是 gVim 显示 C 语言代码的效果。



图 7.5　Vim 的 Logo

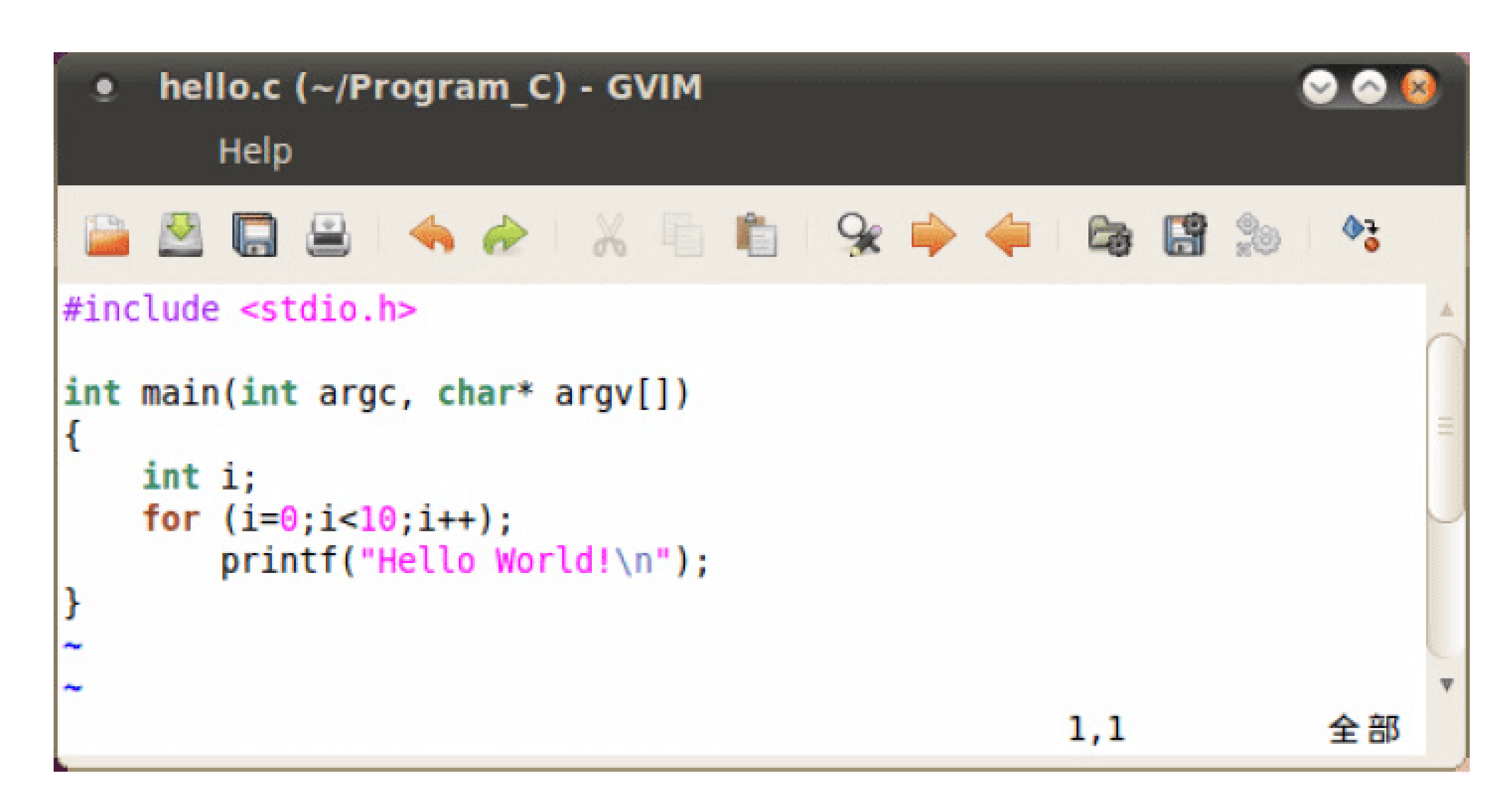


图 7.6　gVim 显示 C 语言代码

【Emacs】

还有一个跟 Vim 同样强大的编辑器，就是 Emacs 了。

Emacs 全称为 Editor MACroS，直接翻译过来就是“宏编辑器”，图 7.7 所示是它的 Logo。Emacs 是一个有着悠久历史的文本编辑器，最初由 Richard Stallman 于 1975 年在 MIT 的时候，协同 Guy Steele 共同完成，比 Vi 的诞生要早一点。这种神一样的编辑器使用了 Emacs Lisp 这种有着极强扩展性的编程语言，从而实现了包括编程、编译乃至网络浏览等功能的扩展。由于扩展功能的强大，所以很多人开玩笑说 Emacs 是一个伪装成编辑器的操作系统。



图 7.7　Emacs 的 Logo

提示：Emacs 还有一个分支，是 1991 年发起的 XEmacs 项目。XEmacs 与 Emacs 有着良好的兼容性，并且对多国语言的处理能力更加强大。它甚至可以在一份文件中同时处理多种不同的语言文字。

如今，依然有很多人在使用 Emacs 编辑器处理着他们的各种工作。在我们 Ubuntu 系统中想用 Emacs 就直接叫来超级牛力安装：

```
$sudo apt-get install emacs
```

安装好之后，就可以直接运行“emacs”命令来启动 Emacs。“emacs”命令本身可以

判断当前的系统环境，如果有图形界面，会启动图形界面的 Emacs，如图 7.8 所示。如果是在纯字符终端运行“emacs”命令，则启动字符界面的 Emacs，如图 7.9 所示。



图 7.8　Emacs 图形界面

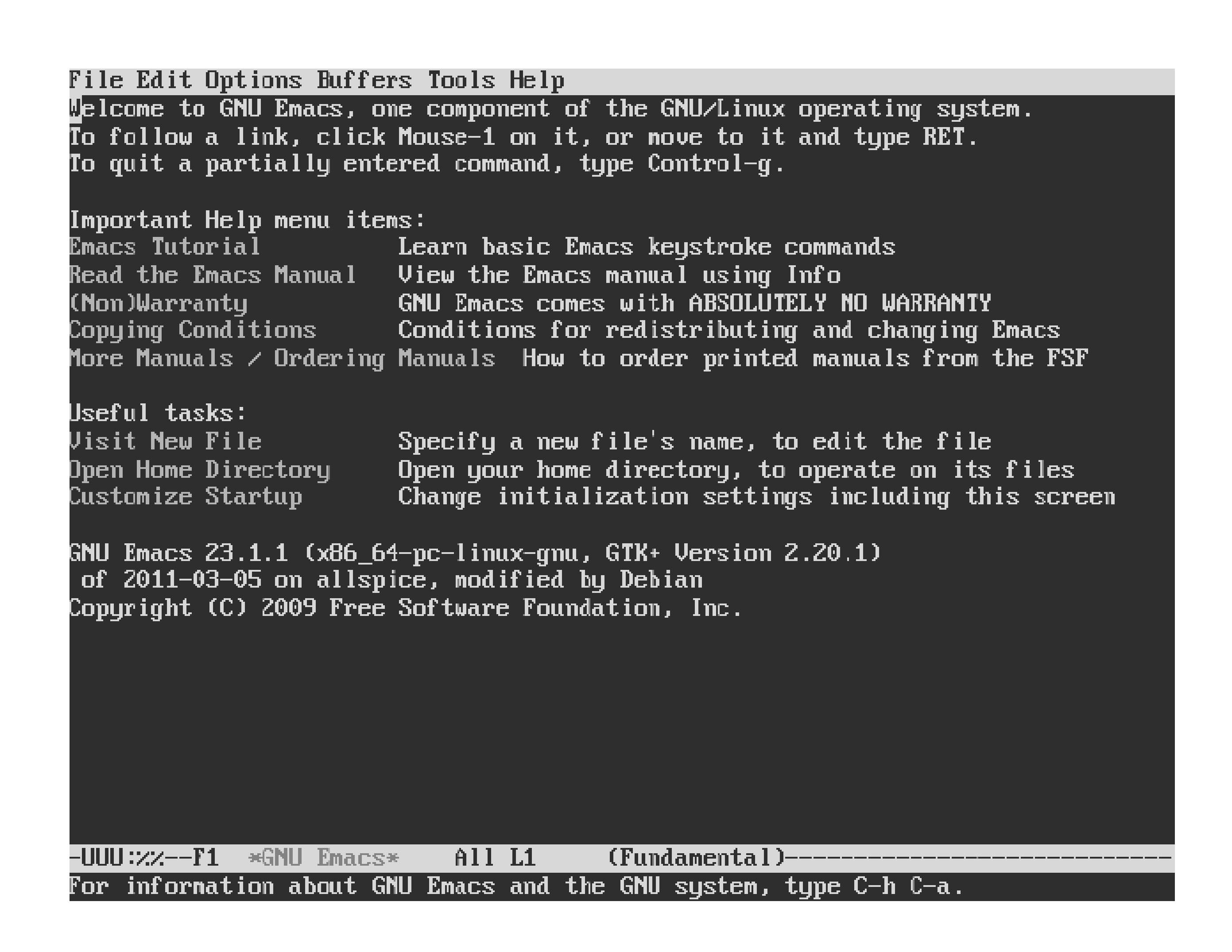


图 7.9　Emacs 字符界面

【最强大的编辑器】

不过有一点要特别注意：千万不要问谁是 Linux 下最强大的文本编辑器！

一直以来，在 Linux 这片自由的天空下，两位公认的顶级的文本编辑器——Vim 和 Emacs，谁也不服对方。两个人都觉得自己才是空前绝后旷古烁今的全能文本编辑器。一旦有谁质疑一下“最强大文本编辑器”的地位，他们两个都会第一时间跳出来，相互指摘对方的缺点，以确立自己在文本编辑器领域的不败地位。

Vim 总是指责 Emacs 说：“那么多的快捷键，记忆起来多麻烦。”

这时候一般 Emacs 会反驳：“你呢？那么多命令难道容易记？”

“初期需要记住的命令确实多一些，”Vim 辩解说，“但是总共就那么几个命令，记

住之后就可以应用自如了。通过简单命令的组合可以实现各种复杂的操作。哪像你，每种操作都有快捷键要记忆，而且还分那么多模式。每个模式都有特定的快捷键，搞得人晕头转向。”

“你还好意思说我模式多？你不也分什么输入模式、指令模式还有行末模式么，搞得新手不知如何是好，连退出都不知道怎么退出。你觉得我模式多？那是我灵活，我功能多。你能看邮件么？你能编写网页么？你能看图片么？我都能，并且还远远不止这些。”

Vim 会冷冷地说：“是啊……所以你才不是最强大的文本编辑器。因为你压根不是文本编辑器，你是个绑定了文本编辑功能的操作系统！”

“胡说，我是编辑器，怎么成了操作系统了？我不是操作系统！你才是操作系统呢，你们全家都是操作系统！”

……

总之呢，一定不要让这两个人遇到一起，更不能在有他们两个的时候提到谁是最好的之类的话题。否则就是：吵不关机死不休！其实要我说，这两者各有特点而已，没什么争吵的必要。您用哪个顺手就用那个就行了。

【经典的 HelloWorld】

介绍了这么多，这会儿懒蜗牛已经完成了他人生中的第一段 Linux 下的 C 语言代码。他写的是一段最简单、最经典的 C 语言代码——HelloWorld。程序全文如下：

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    printf("HelloWorld!\n");
}
```

简洁明了吧。这段程序也没什么实际意义，只是试验一下 C 语言的编译而已。懒蜗牛同学写好了这个程序之后，把它保存在了/home/lanwoniu/Program_C 目录下。这是他刚刚建立的一个目录，把这个源代码存为了 hello.c 文件。然后，就准备进行编译了。

提示：建立目录使用命令：mkdir <路径>/<目录名>。

7.1.3 编译和运行

【编译 C 语言】

编译很简单，找来编译器 GCC 就可以了。只见懒蜗牛同学运行：

```
$cd /home/lanwoniu/Program_C
```

进入这个目录里，然后：

```
$gcc ./hello.c
```

这样就编译完了，简单吧。运行完了之后，编译出的结果会被命名为一个 a.out 文件，就放在当前目录下，懒蜗牛执行了 ls 命令一下，果然看到了这个文件：

```
$ls
a.out hello.c
```

这个 a.out 就是编译出的二进制文件，赶紧执行试试：

```
$./a.out
HelloWorld!
```

嗯，果然如预期所料。

咱刚才说了，GCC 这个名字已经并不单单指一个编译器了，而是很多种语言的编译器的组合。除了编译 C 语言的“gcc”命令之外，另外一个最常用的就是用于编译 C++程序的“g++”命令了。

【C++语言简介】

C++，这个词在懒蜗牛同学所在的这个国家通常被读做“C 加加”，同理，地球对面的程序员通常读做“C plus plus”。它是一种使用非常广泛的计算机编程语言。它完全兼容 C 语言，在 C 语言的基础上增加了对对象的支持。

早在 20 世纪 80 年代，贝尔实验室的本贾尼 • 斯特劳斯特卢普（Bjarne Stroustrup）博士最初发明 C++语言的时候，将它命名为“C with Classes”，直译过来也就是“带有类概念的 C 语言”。C++最初是作为 C 语言的增强版出现的，但随着它的发展，从增加类开始，不断地增加新特性。什么虚函数（virtual function）、运算符重载（operator overloading）、多重继承（multiple inheritance）、模板（template）、异常（exception）、RTTI、命名空间（name space）逐渐被加入标准。

到 1998 年，国际标准组织（ISO）颁布了 C++程序设计语言的国际标准 ISO/IEC 14882-1998。然而由于 C++语言被设计得太复杂、太高深、太科幻，以至于到现在为止，都没有一个编译器能够 100%地支持这个标准。不过也不用担心，绝大多数编译器的绝大多数行为还是一致的。

【C++的编译】

懒蜗牛很快又试了试“g++”命令的使用。要使用 g++，首先得有段 C++语言的程序啊，其实“g++”是完全可以编译 C 程序的，不过为了让程序更 C++一些，懒蜗牛同学还是把程序改了改，写出了 C++版本的 HelloWorld：

```
#include <iostream>

int main(int argc, char* argv[])
{
    Std::cout << "Hello World C++!"<<std::endl;
}
```

修改之后，懒蜗牛将程序另存为 hello.cpp 文件并运行：

```
$g++ ./hello.cpp
```

同样也编译出了 a.out 文件，运行效果和刚才那段 C 语言的程序没啥区别。

提示：编译输出的文件如果有同名文件在目录下，编译器将直接覆盖原有文件，不做任何提示或询问。

7.1.4　C/C++语言集成开发环境

咱们已经说明白了，这个 GCC 只是个编译器，不是集成开发环境。但是并不是说我

们 Ubuntu 系统里就没有 C/C++语言的集成开发环境。毕竟有时候，有的用户，还是用集成开发环境更顺手一些。那我就介绍几个常见的吧。

【Anjuta】

- ❑ 姓名：Anjuta
- ❑ 性别：无
- ❑ 毕业院校：开源社区
- ❑ 源码面貌：由 C 语言编写
- ❑ 主要工作技能：C 语言开发，C++语言开发
- ❑ 次要工作技能：Java 语言、Python 语言项目开发
- ❑ 联系地址：http://www.anjuta.org/index.html

这位 Anjuta 是一个专门用来开发 C\C++程序的集成开发环境，也可以用来开发 Java 和 Python。他主要用于 Gnome 桌面环境中，尤其开发 Gnome 程序或基于 GTK+库的程序，较为方便灵活。并且这个软件身材小巧，节约资源，属于经济实惠型的软件，如图 7.10 所示。

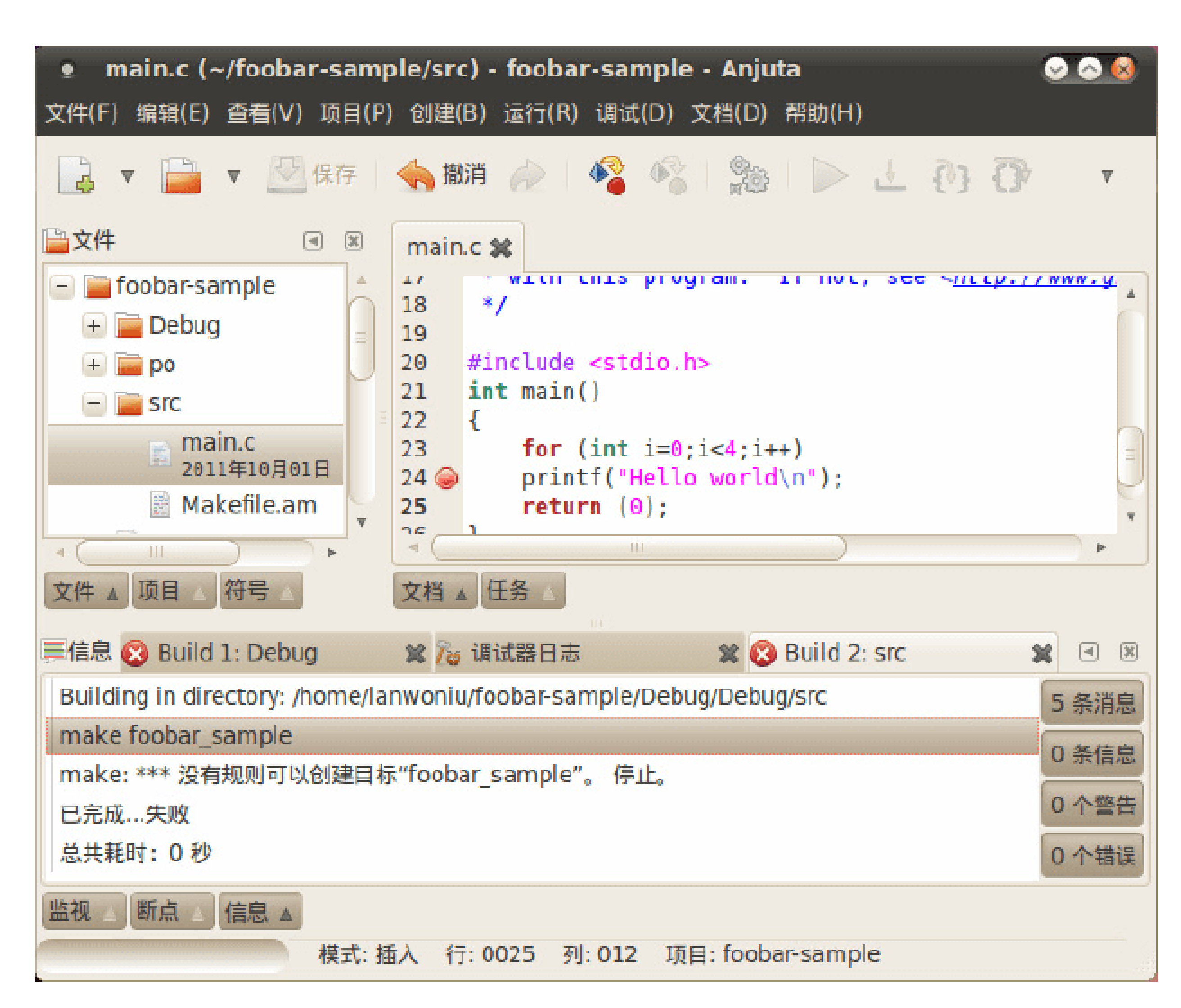


图 7.10 Anjuta 集成开发环境

【Geany】

- ❑ 姓名：Geany
- ❑ 性别：未知
- ❑ 毕业院校：开源社区
- ❑ 源码面貌：由 C 语言编写
- ❑ 主要工作技能：C 语言开发、Java 语言开发、PHP 语言开发
- ❑ 次要工作技能：HTML、Python、Perl、Pascal、Haskell、LaTex 等语言开发
- ❑ 联系地址：http://www.geany.org/

Geany 是一个使用 gtk+工具包编写的软件。他的功能介于文本编辑器（比如 gedit）和集成开发环境（也就是 IDE，比如 Anjuta）之间，可以算是一个“有集成开发环境基本特

性的文本编辑器”。他的体积同样小巧，并且依赖的软件包比较少。同时，Geany 还支持许多种程序设计语言，是一个全功能的编辑器，图 7.11 所示是 Geany 的工作界面。

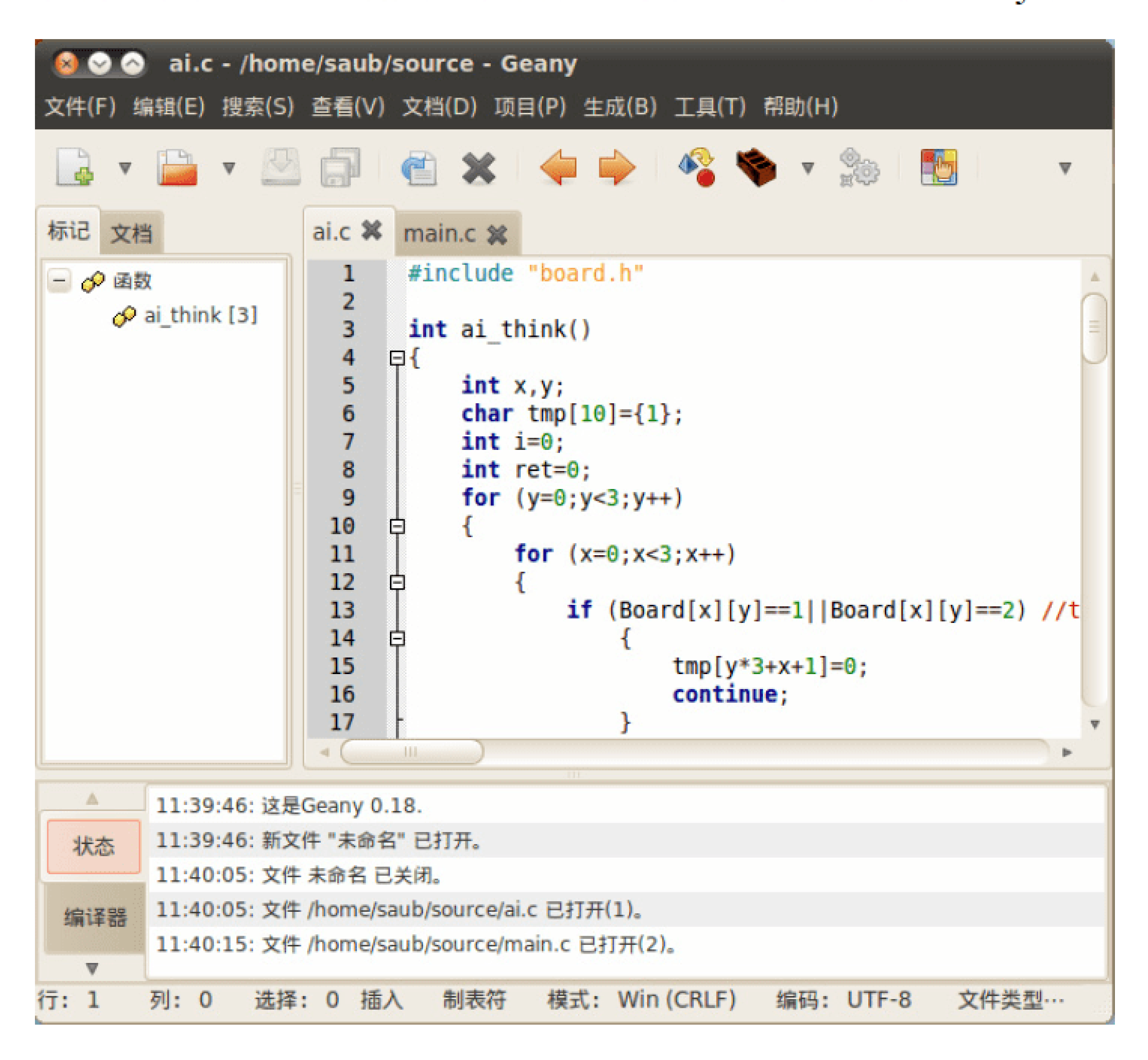


图 7.11　Geany 集成开发环境

【KDevelop】

- ❑ 姓名：KDevelop
- ❑ 性别：中
- ❑ 毕业院校：开源社区，KDE 项目
- ❑ 源码面貌：由 C、C++语言编写
- ❑ 主要工作技能：C、C++、Perl、Python、PHP、Java、Fortran、Ruby、Ada、Pascal、SQL，以及 Bash 脚本程序的编写
- ❑ 次要工作技能：主要技能里的那些还不够你用的吗
- ❑ 联系地址：http://kdevelop.org/

KDevelop 是 KDE 环境下的集成开发环境，功能比较强大，从图 7.12 中可以看出一些。如果你用 KDE 桌面环境，或者要开发基于 KDE 环境，基于 Qt 库的程序，那么向你强烈推荐这个集成开发环境。他还能够支持 CVS、Subversion、ClearCase 和 Perforce 这些版本控制系统，为多人共同开发提供了便利。

提示： KDevelop 不在 Ubuntu 的软件源中。如果要求安装，需要添加 ppa 源。运行以下命令：

```
$sudo add-apt-repository ppa:kubuntu-ppa/backports
$sudo apt-get update
```

之后即可通过 apt 安装 kdevelop。

```
$sudo apt-get install kdevelop
```

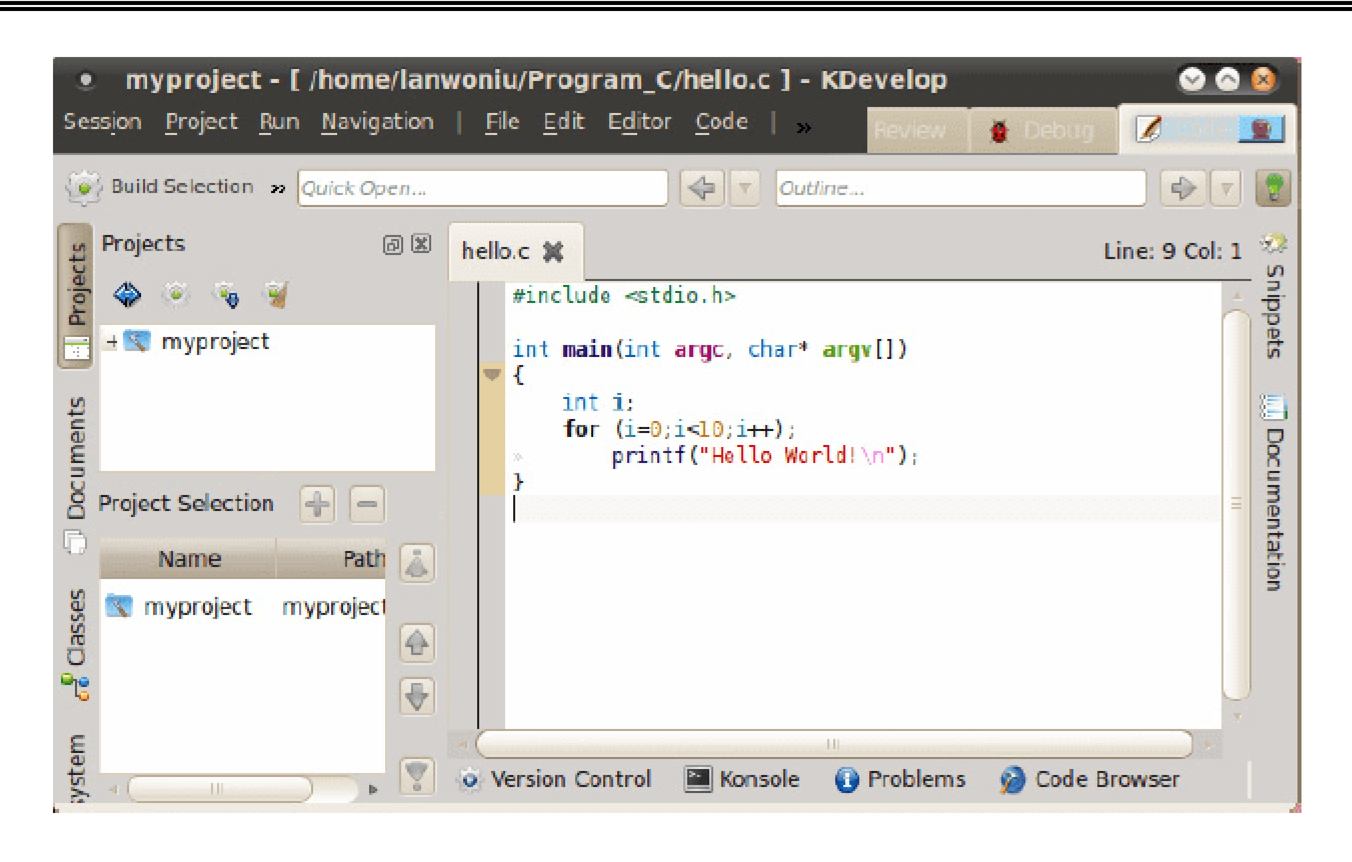


图 7.12　KDevelop 集成开发环境

【Eclipse】

- ❑ 姓名：Eclipse
- ❑ 性别：为啥老有这项
- ❑ 毕业院校：IBM 公司、Eclipse 基金会
- ❑ 源码面貌：由 Java 语言编写
- ❑ 主要工作技能：Java 语言开发
- ❑ 次要工作技能：C\C++语言项目开发，PHP 语言项目开发
- ❑ 联系地址：http://www.eclipse.org/

这个 Eclipse 大概对于很多开发 Java 的同志们来说已经非常熟悉。不过既然它已经落在了开源社区手里，就跑不掉被热心的开发者插满插件的命运。其中就有支持 C\C++语言的插件 CDT。图 7.13 所示是 Eclipse 的启动界面。



图 7.13　Eclipse 启动界面

提示：2001 年 11 月 IBM 公司将 Eclipse 贡献给开源社区。

相比上面两个集成开发环境，Eclipse 更加商业化，更加专业一点，毕竟有 IBM 公司雄厚的技术底蕴做基础，写出来的东西确实相对好用些。不过由于是用 Java 写出来的这么大一个程序，所以运行起来的速度可能就不那么理想了。

7.2　PHP 开发环境的搭建

搭建好了 C 语言开发环境后，懒蜗牛同学又开始着手试着研究 PHP 语言了。这主要是因为他觉得 C 语言还是有点难度，好像 PHP 更容易入门些。于是他就开始了搭建 PHP 开发环境的工作。

7.2.1　PHP 是个神马

PHP（Hypertext Preprocessor）是一种脚本语言，主要用于处理动态网页。不过它也包含了命令行运行接口，甚至还能产生拥有图形用户界面（GUI）的程序。PHP 最早由拉斯姆斯·勒多夫（Rasmus Lerdorf）在 1995 年发明，就是图 7.14 中所示的这位仁兄。要说起 PHP 的作用和意义，那就得从很久以前说起了……



图 7.14　拉斯姆斯·勒多夫

【静态的 HTML】

话说很久很久以前，互联网才刚刚兴起。那时候网页上使用的是 HTML 语言，也就是超文本标记语言——HyperText Markup Language 来实现排版。似乎是带“超”字的东西都比较强大而流行，比如超人、超市、超女之类的，于是网页这种形式的媒介很快流行起来（这都挨得着么……）。用户的浏览器下载下来用 HTML 语言写的网页，然后按照 HTML 语言的规则解释为一张调整好各种版式、字体、图片等内容的网页，如图 7.15 所示。

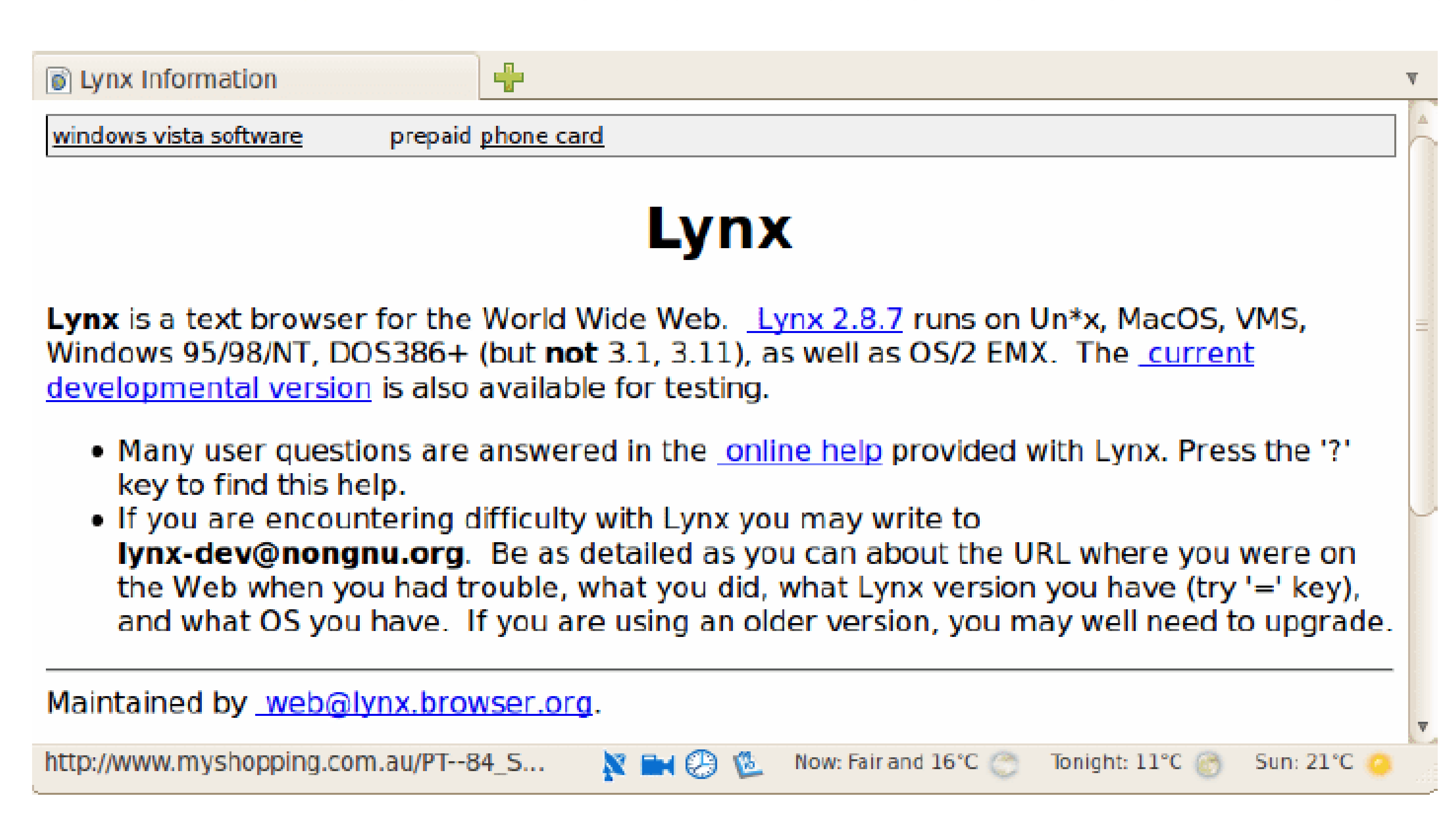


图 7.15　静态网页

【在客户端执行的脚本】

不过过了几年，人们就不满足于仅仅靠 HTML 提供的这种静态的网页了。于是各种网页脚本语言开始盛行，比如 JavaScript。这种脚本嵌入在 HTML 的网页源码中，用户的浏览器下载下网页的源码之后，除了按照 HTML 排版出网页以外，还要运行网页上的 JavaScript 语言写的脚本程序。有了这样的脚本语言，就可以实现很多有意思的效果，比如图片的移动（比如某些网页上飘来飘去的广告），背景的切换，甚至实现一个网页上的小游戏，都没有问题。于是，网页开始越来越有意思了。图 7.16 所示就是一个有一定互动性的动态网页的例子。



图 7.16　互动网页

【在服务器端执行的脚本】

不过这种脚本语言也有它的问题。随着这种脚本运用得越来越广泛，写得越来越复杂，执行的效率就难以保障了。用户的电脑多种多样，有的是奔腾 166 MHz 的 CPU，16 MB 的内存；有的是 8 核 CPU，8 GB 的内存。这两台电脑如果运行同一个网页行的脚本，速度肯定不一样。并且由于不同的浏览器对脚本的解释也会有些细微的差别，因此一个脚本要想在各种电脑、各种浏览器上都能有相同的效果，需要做大量的兼容性测试工作。

这时候，PHP 语言吸引了人们的目光。PHP 也是一种脚本语言，但它是运行在服务器端的。它同样可以提供一些互动性的或者动态的网页，但这种动态和互动性是依赖服务器的运算能力来实现的，不需要用户的浏览器和电脑做过多的工作。

这样做的好处就是减少了客户端浏览器和计算机硬件的不同带来的差异性，减轻了客户端浏览器的工作压力，让网页的效果更有保证。

7.2.2　解释型语言

PHP 是一种解释型的语言，跟刚才咱们说的编译型的 C 语言不一样。那么解释型语言又是怎么回事呢？

解释型语言就是由一个统一的解释器根据脚本程序进行各种操作，实现各种各样的功

能。假设还是你懒得翻书这件事。如果用编译型语言，就相当于设计出一张“自动翻书机”的图纸，然后照着图纸制造出这么个机器，之后你就可以懒到看书不用翻了。

如果用解释型语言呢？那你需要先有一台“全功能懒人助手机器人”，然后你写一段翻书的程序输入到里面，程序里写明了该怎么翻、如何翻、全是字的页隔多长时间翻、有插图的隔多长时间翻等。然后这个全能机器人理解了你写的程序后，就可以帮你做翻书的动作了。

这时候，你写的那个清单，就相当于解释型语言的程序清单。而全能机器人能够解释你的程序，并做出相应的动作，于是它就是你这种“翻书语言”的解释器了。

从我们操作系统的角度看，编译型语言是通过编译器创造出新的程序，然后让这个新的程序去完成你需要的任务；而解释型语言是有一个通用的，干什么都行的解释器程序（比如全能机器人），通过写程序告诉这个程序该去干什么，然后由解释器程序去具体实现各种功能。

提示：我们熟悉的 bash 其实也是一种解释器，Shell 脚本就是用 bash 能够看懂的语言写的一段程序。

7.2.3　安装 Apache 和 PHP

懒蜗牛同学准备安装 Apache 了。

【安装必要的软件包】

有人问，这个 Apache 是什么？不是在说 PHP 么？你是不是印错了？告诉您，没错。这个 Apache 是我们 Linux 下的一个 Web 服务器软件，对外提供 Web 服务。再简单点说吧，有了 Apache，就可以把你的电脑做成一个网站了。Apache 的 Logo 是一片彩色的羽毛，您可能曾经看到过，如图 7.17 所示：

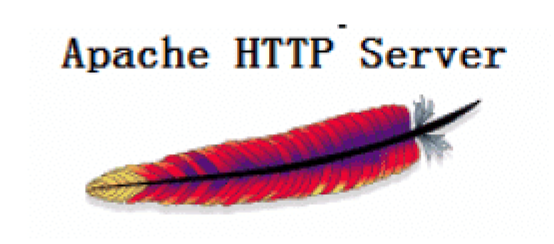


图 7.17　Apache 的 Logo

因为 PHP 主要是用在网页上的脚本语言，而且还是在服务器端执行的，所以当然先得有个 Web 服务器才好使用 PHP 脚本。

提示：一些轻量级的 Web 服务器软件，例如 Lighttpd、thttpd 等，也可以支持 PHP 脚本。

安装 Apache 还是找超级牛力，PHP 也找超级牛力装，为了省事，可以写在一条命令里，类似这样：

```
$sudo apt-get install apache2 php5-mysql
```

【验证 Apache 正常】

安装的过程没什么悬念，安装后你就可以用 Firefox 访问 http://127.0.0.1/这个地址来看看效果了，如果正常，应该会看到如图 7.18 所示的页面。

提示：根据 TCP/IP 协议，127.0.0.1 为代表本机的特殊地址。

那么这个简陋的页面是从哪里来的呢？

装上了 Apache 之后，你的电脑就相当于一个网站的服务器了。而这个页面，就相当

于网站的首页。这个页面存储在/var/www/目录下，里面有个 index.html 文件，里面的内容很简单，大约就是这样：

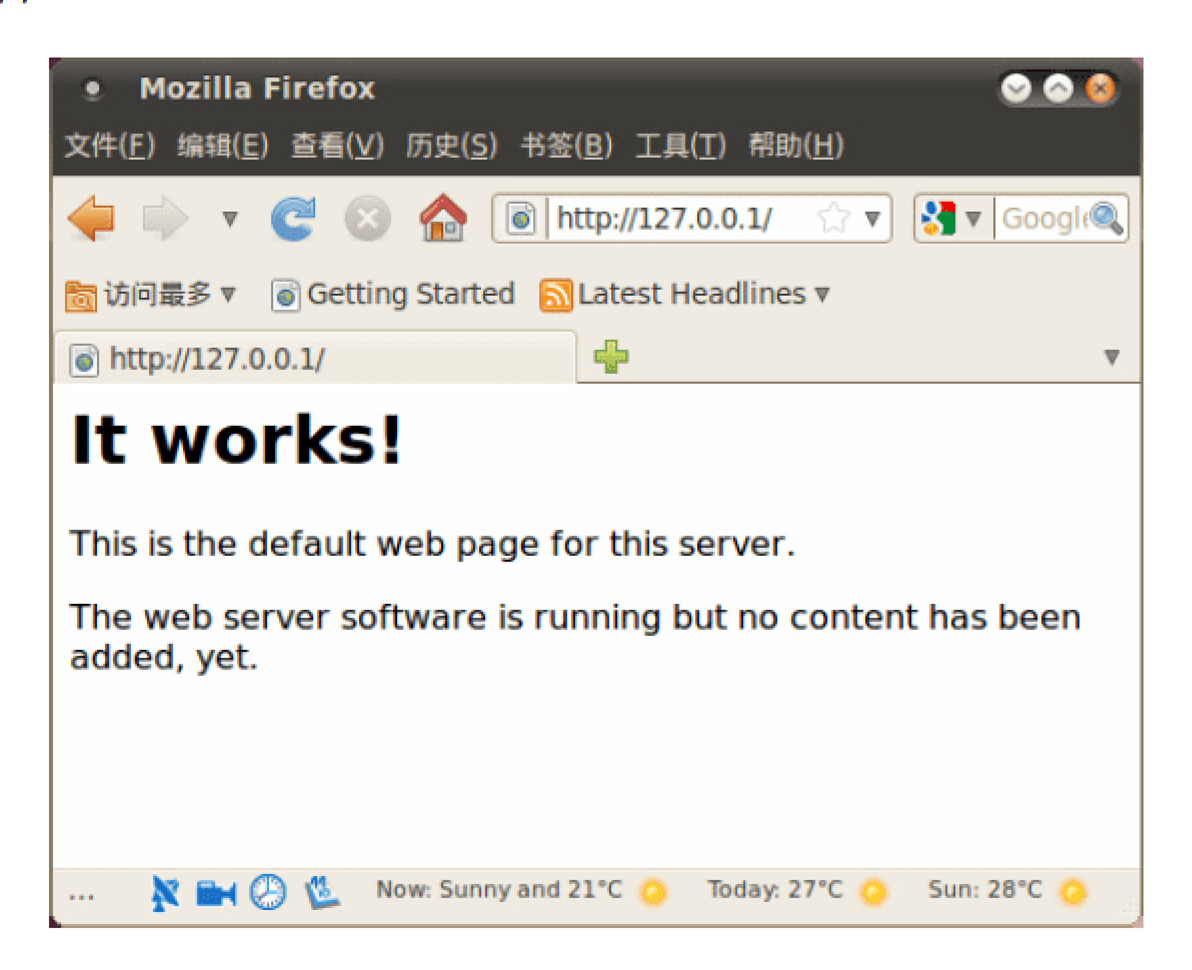


图 7.18　Apache 默认页面

```
<html><body><h1>It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
</body></html>
```

用狐狸妹妹访问 127.0.0.1 的时候，狐狸妹妹就连接到本机的 80 端口，然后会发现 Apache 在那里笑盈盈地等着，并把这个简单的 index.html 文件交给狐狸妹妹，狐狸再根据 HTML 语言的规则把文件显示出来，就是您刚才看到的那个页面。

如果能够显示这个页面，Apache 算是正常工作了，但我们还没有看到 PHP 的身影呢。

【验证 PHP 正常】

要验证 PHP 是否正常工作，需要写一个 PHP 的页面。还在刚才那个/var/www/目录里，咱们再写一个 index.php。当然，这个目录不是随便哪个人都能写入的，需要 sudo：

```
$sudo gedit /var/www/index.php
```

这样，gedit 会打开一个空文件，然后我们往里面写入如下内容。暂时照着写，不要问为什么。

```
<html><body><h1>PHP works!</h1>
<p>This is the used to test PHP.</p>
<?php
phpinfo();
?>
</body></html>
```

写完之后保存好，再访问 http://127.0.0.1/index.php（如果访问 http://127.0.0.1/，还会获取到 index.html），然后，就是见证奇迹的时刻，如图 7.19 所示。

我们写下的短短几行的代码竟然显示出这么复杂的一个页面。这个页面里介绍了当前正在运行的 PHP 的一些信息，比如版本、各种相关的配置等。看到这个页面，就说明 PHP 已经正常地运行起来了。

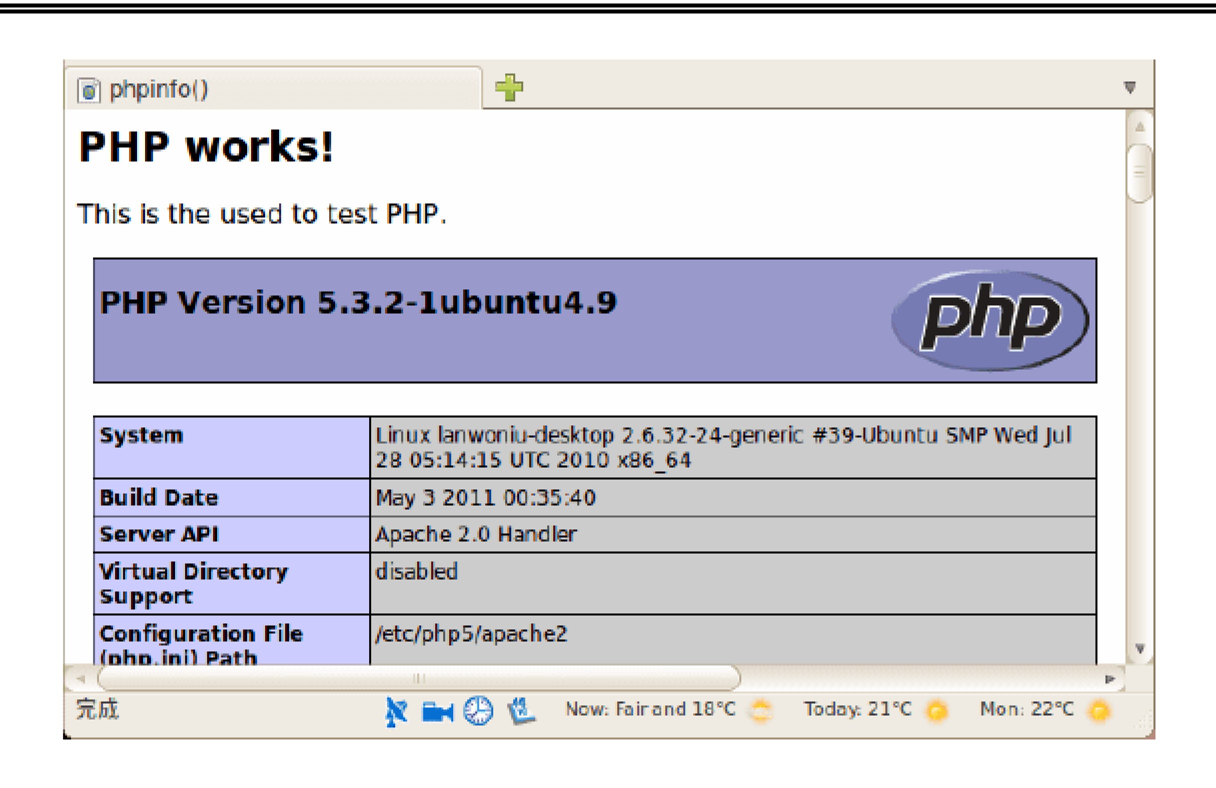


图 7.19　PHP 信息页面

7.2.4　又是 HelloWorld

好了，现在开发 PHP 的环境装好了，懒蜗牛同学要开始学习怎么写 PHP 程序了。

【浏览器中的 PHP 页面】

刚才我们看到了，PHP 的程序是嵌入在 HTML 语言之间的，用标记“<?php”作为开始，“?>”作为结束。这两个标记之间的内容，就是 PHP 的语句。懒蜗牛同学又写了个经典的 HelloWorld 程序，存为 test.php 文件，放在了/var/www/目录下。程序全文如下：

```
<html><body><h1>My PHP test</h1>
<?php
printf("<B>Hello World!</B>");
?>
</body></html>
```

提示：曾经 PHP 语言的开始标记也可简写为“<?”，但为了防止和其他的标记符号冲突，PHP 官方现在推荐使用标准的“<?php”。

这样，当懒蜗牛同学用狐狸妹妹访问 http://127.0.0.1/test.php 的时候，就是下面这样一个过程。

（1）首先，Apache 接到狐狸的请求，知道她想要查看 test.php 这个文件，于是就把文件拿过来。但是根据扩展名发现这是一个.php 文件，于是就叫来 PHP 处理。

（2）然后，PHP 就查找这个文件中所有“<?php”和“?>”之间的内容，并且按照 PHP 语言去解释它们，执行相应的操作，用操作输出的结果替换掉“<?php”和“?>”之间的内容，再交回 Apache 手里。这时候的代码里就没有 PHP 的内容了，比如懒蜗牛同学写的这个简单的 test.php 页面，经过 PHP 翻译之后，大约就是下面的样子：

```
<html><body><h1>My PHP test</h1>
<B>Hello World!</B>
</body></html>
```

（3）之后，Apache 才将 PHP 翻译过的这个文件交给狐狸妹妹，让她去根据 HTML 语言的规范去显示页面。狐狸显示出来的 test.php 大约就是图 7.20 所示的这个样子了。所以说狐狸根本看不到什么 PHP 语句，这就是所谓的“运行在服务器端”的脚本语言。

【命令行中的 PHP 脚本】



图 7.20　test.php 效果

另外，PHP 脚本也不一定非要用在网页上，如果你对 PHP 语言非常熟悉，想用它写个脚本实现一些简单的功能，比如批量处理文件、自动格式化硬盘之类的，也是没问题的。不过这需要一个命令行下的 PHP 解释器，需要安装“php5cli”这个软件包，运行以下命令即可，都是老熟人了，我就不讲解这个命令是什么意思了：

```
$sudo apt-get install php5cli
```

装好之后，你就可以写 PHP 脚本了，比如下面这个简单的例子：

```
#!/usr/bin/php
<?php
$cpuinfo=system("cat /proc/cpuinfo");
printf ($cpuinfo);
?>
```

这个文件第一行，咱们之前说过，说明了这个脚本的解释器是/usr/bin/php。下面就是 PHP 的语句了，当然，虽然不是网页，但 PHP 语句也一定要写在<?php 和?>之间。把这个文件写好了之后，存储为 cpu.sh 并赋予执行权限之后，即可执行这个脚本看效果了，其实就是用来查看 CPU 信息的一个小脚本。

提示：使用 chmod +x cpu.sh 命令来赋予可执行权限。

7.3　Java 开发环境的搭建

除了 C 语言外，应用最广泛的语言大概就是 Java 了，尤其是在开发手机应用方面，更是 Java 的天下。因此，懒蜗牛同学决定，还得好好学一下 Java。

7.3.1　半编译型语言

Java 也是一种电脑编程语言，拥有跨平台、面向对象、泛型编程等特性。它最初被命名为 Oak（橡树），当时的目标是用于电视机、电话、闹钟、烤面包机等家用电器的控制和通信。可是由于这些智能化家电的市场需求没有预期的高，于是 Oak 被改造为应用在互联网上的编程语言。1995 年 5 月，它被正式命名为 Java，并伴随着互联网的迅猛发展而发展，逐渐成为重要的网络编程语言。它的 Logo 大概您不会陌生，就是图 7.21 所示的这杯咖啡。



图 7.21　Java 的 Logo

前面说了，这个编程语言有编译型的，有解释型的。那好，提问：Java 是哪种类型的语言呢？

- ❑ A．编译型

- ❑ B．解释型
- ❑ C．以上皆错
- ❑ D．人妖型

正确答案：C、D（我说是单选了吗？）。

其实，Java 既不是编译型，也不是解释型。它有它的中性美，是多少有点半男不女、半上不下、半人半妖的这么一种人妖型语言。好吧，我承认这个名字不那么好听，那么我们就管它叫做半编译型语言吧。为什么这么叫呢？

【既需要编译器，又需要解释器】

首先，跟编译型语言一样，要想运行 Java 的源程序，必须要经过编译的步骤。但是 Java 程序编译出来之后并不能像 C/C++语言一样直接在机器上运行。也就是说，Java 编译器编译出来的并不是一个符合当前系统的二进制格式的程序，而是一种特殊结构的二进制程序。

那么这种特殊的程序怎么运行呢？要想运行这个程序，就和解释型语言一样，需要一个解释器。Java 的解释器就是 Java 虚拟机。一般如果要运行一个 Java 写的软件，就必须装一个叫做 JRE 的东西。他负责给 Java 程序创造出一个可以自由运行的空间，在那里面，这个编译好的 Java 程序才能运行。

就好像，我们宽敞的内存是一片大草原，GCC 编译出来的程序就是各种哺乳动物，什么牛、羊、猎狗之类的。而 Java 编译出来的程序就是一堆鲤鱼、黄鳝、泥鳅之类的东西。那么 JRE 的工作，就是挖一个水塘，或者做一个鱼缸。

【Java 的特点】

那为什么 Java 要整得这么繁琐呢？又需要编译器，又需要解释器，有什么好处？好处很大，最大的好处就是跨平台。

作为编译型语言，编译出来的程序跟目标平台是紧密相关的。平台包括硬件和操作系统。也就是说，同样一套程序的源代码，如果要在不同的操作系统或者不同的硬件平台上运行，那么就需要用不同的编译器，针对每一种系统和硬件分别进行编译，才能够运行。

而 Java 就可以解决这种困境。它可以只编译一次，就能把编译出的二进制程序放到不同的平台上运行。当然，前提是那些平台上都安装了 Java 的解释器。各个平台的差异性由 Java 的解释器统一处理掉了，编写 Java 程序的人不需要考虑程序是运行在哪个系统，哪种 CPU 中，只要程序写出来，编译完了，就可以放到任何地方运行。Java 的理想就是：Write once，run anywhere。

🔔**提示：** 实质上，由于各个系统的差异性，Java 编译出的程序还是需要针对特定平台进行一定的调试和修改。所以有的程序员戏称 Java 是“Write once,debug anywhere”。

当然，解释型语言照样可以实现跨平台，只要不同的平台上都有相应的解释器就可以。比如 bash 其实就可以算是一种解释型语言，同样一个脚本，不管你是在 Linux 还是 BSD 系统，或者 Windows 的 cygwin 模拟环境下，都可以实现相同的功能。但是与之相比，由于经过了编译，Java 的二进制程序更接近汇编指令，因此翻译起来效率比纯解释型语言高很多。

7.3.2 JDK 和 JRE

作为一种编译型语言，要编译程序，就需要编译器。

作为一种解释型语言，要运行程序，就需要解释器。

而作为一种人妖型语言的 Java，既需要编译器，又需要解释器。所以，我们就来说说怎么装这俩东西。

【安装开源的 JDK 和 JRE】

如果你想要运行一个 Java 的程序，比如咱们之前介绍过的 BT 下载软件毒蛙，那么你的系统里面就必须有解释器——JRE。JRE 就是 Java Run Time 的缩写，只要你想运行 Java 程序，无论在哪个系统上，都得安装它。而如果你想开发 Java 程序，那就不止需要 JRE 了，还需要 JDK，也就是 Java Development Kit，Java 开发工具。

这两种东西在我们的软件源里有不止一种。您可以在新立得里面搜索一下 JRE，会有不少结果。其中有一个 openjdk 系列的软件包，包括了编译器 JDK 和解释器 JRE。看名字就能明白，这是一套 Java 的开源实现，如果您只是想运行些简单的 Java 程序，基本上装这个包也就可以了。不过既然懒蜗牛同学是想做 Java 的开发，那还是用更权威的比较好，那就是 sun-java6。

【安装权威的 JDK 和 JRE】

sun-java6 这一套开发套件默认并不在源中，需要稍微操作一下。别急，一点儿也不难。

（1）首先用任何顺手的编辑器打开/etc/apt/sources.list 文件，例如用 gedit，就这样：

```
$sudo gedit /etc/apt/sources.list
```

（2）打开 sources.list 文件之后，在文件的最后加上这样一行：

```
deb http://archive.canonical.com/ lucid partner
```

（3）记得我们说过，更改 sources.list 文件之后一定要更新，所以：

```
$sudo apt-get update
```

好了，这样就可以直接用超级牛力来安装 Java 相关的东西了。比如安装 JRE 只要运行：

```
$sudo apt-get install sun-java6-jre
```

就安装好了。或者用新立得安装 sun-java6-jre 这个软件包，也一样。如果需要做 Java 开发，就安装 JDK。

```
$sudo apt-get install sun-java6-jdk
```

7.3.3 再说 Eclipse

理论上有了 JDK 之后，就可以用任何一个文本编辑器来编写 Java 程序了，就像之前我们的懒蜗牛同学编写 C 程序那样。不过多数人不会这么开发 Java，因为实在是有点困难。多数人还是会选择用 Java 的集成开发环境，Eclipse 就是个不错的选择。上次我们介绍 Eclipse 用来作为 C/C++的开发环境那只是个客串，这回开发 Java，就是 Eclipse 的主场了。

好，下面详细介绍一下。

Eclipse 这家伙长得有些晦气，别人看了他总有种世界末日的感觉（日食嘛）。他本身就是一个 Java 程序，最初是由 IBM 公司开发的。而他的功能呢，是作为一套集成开发环境，来开发 Java 程序，于是总给人一点“煮豆燃豆萁”的感觉。行了，废话不多说了，软件源里就有 Eclipse，可以直接叫超级牛力来装：

```
$sudo apt-get install eclipse
```

不过源里的 Eclipse 版本一般比较旧，如果想用最新的版本，还是到官方下载比较好：

```
http://www.eclipse.org/downloads/
```

下载那个“Eclipse IDE for Java Developers”。下载下来应该是一个 TAR 包，直接解开就可以了，不用安装，绿色无污染，没有三氯氰胺，没有苏丹红，没有地沟油，懒蜗牛同学就在用它。

7.3.4　还是 HelloWorld

安装好了 JDK，准备好了 Eclipse，懒蜗牛同学又要写 HelloWorld 了。

【首次启动 Eclipse】

首先，懒蜗牛运行了刚刚下载来的 Eclipse。Eclipse 不紧不慢地显示出了如图 7.22 所示这么个窗口问懒蜗牛：你的工作目录设在哪里呢？

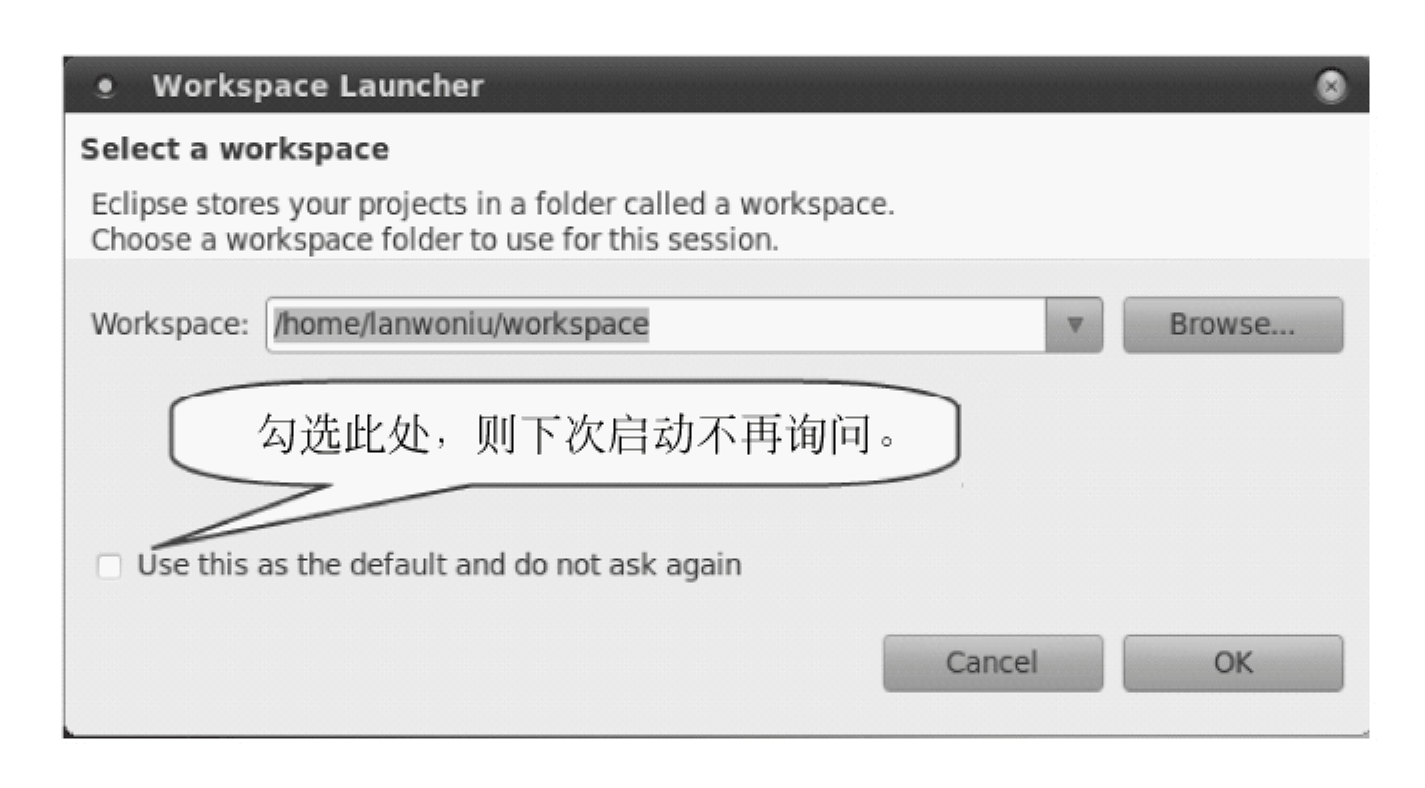


图 7.22　Eclipse 询问 Workspace 路径

这个工作目录（Workspace）就是说，以后用 Eclipse 创建的各种项目、源文件什么的就都会存到这个目录中（当然，Eclipse 不会乱存，会在里面分门别类建好目录的）。一般放在用户的家目录下就可以了，懒蜗牛就没有改动，使用了默认的路径，并且还勾选了下边那个“Use this as the default and do not ask again”复选框。这个复选框的意思就是：“以后都这么办，别再问了。”勾选了之后，懒蜗牛单击了 OK 按钮。

之后，由于是第一次启动，Eclipse 象征性地表示了一下欢迎，显示出了欢迎界面，如图 7.23 所示。

不过懒蜗牛同学并没有跟他客套，直接单击了右边的“Workbench”按钮，于是 Eclipse 就进入了工作状态，如图 7.24 所示。



图 7.23 Eclipse 欢迎页面

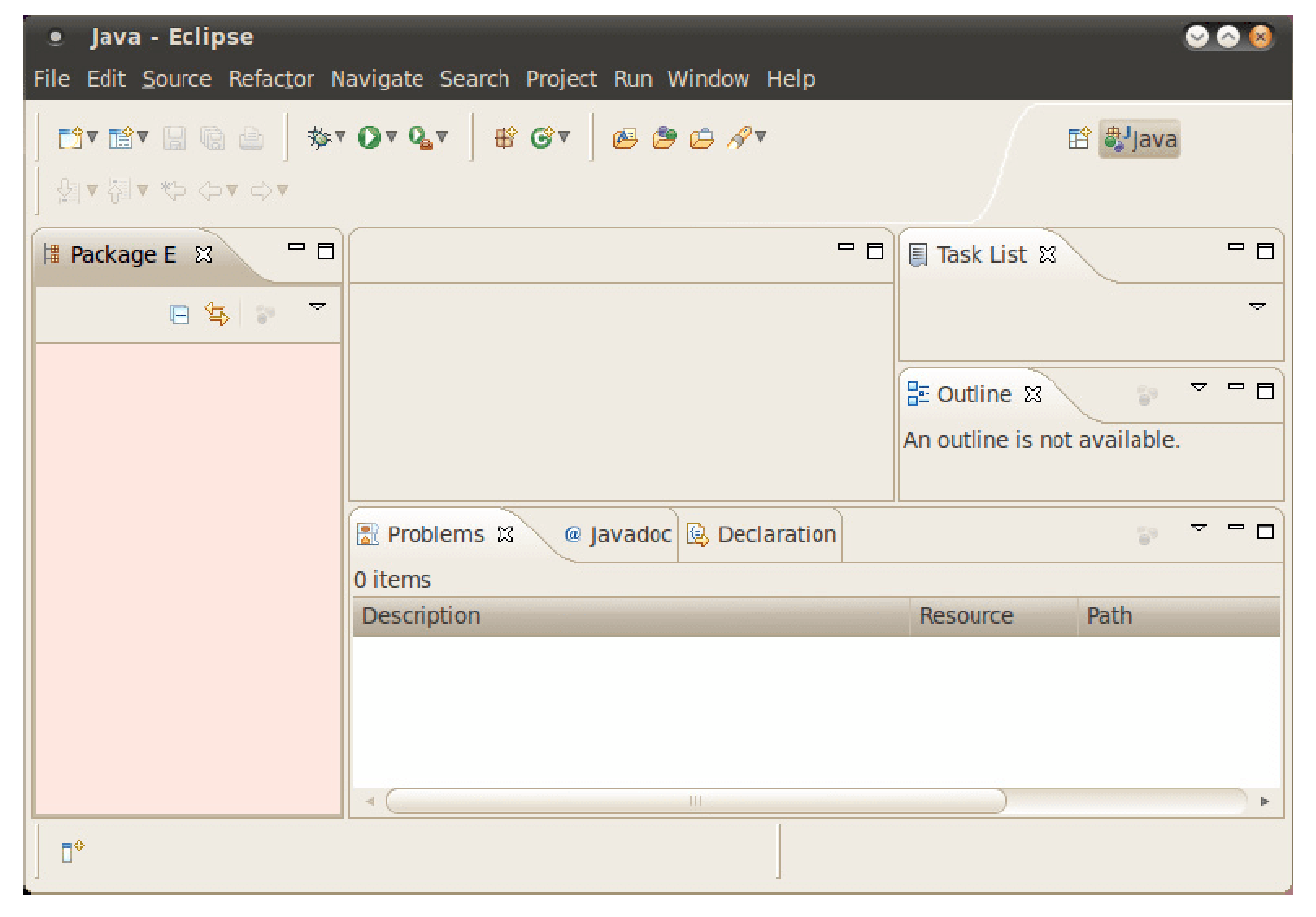


图 7.24 Eclipse Workbench

【创建 Java 项目】

现在界面上自然什么都没有。于是懒蜗牛同学选择了 File|New|Java Project 级联菜单，这就是要创建一个新的 Java 项目了。于是，Eclipse 弹出了创建项目的对话框，如图 7.25 所示。

在这个对话框里，懒蜗牛只是在 Project name 文本框中写下了这个项目的名字，叫做 testjava。其他的内容都没有改变，就单击了 Next 按钮。

在之后的如图 7.26 所示的这个页面，可以对要创建的工程进行一些设置。不过懒蜗牛同学还是新手，不知道该设置些什么，就直接单击了 Finish 按钮，于是，一个工程就建好了。

【创建包】

工程建出来之后就回到了 Wordbenck 界面，这时候在左侧的 Package Explorer 标签中就可以看到这个项目了，不过目前这个项目下面除了有一个名叫 src 的空目以外，啥也没有，如图 7.27 所示。

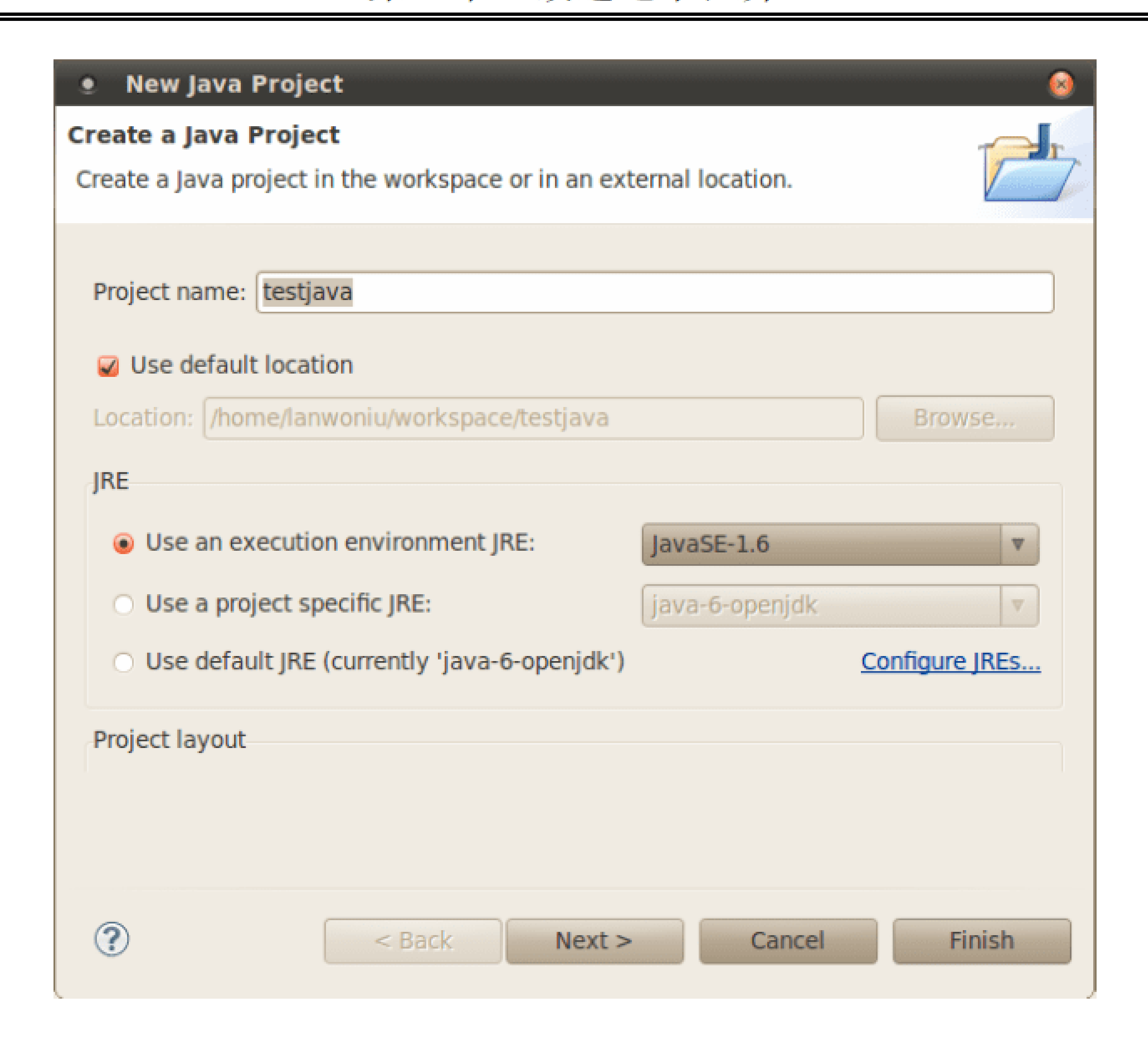


图 7.25　创建项目页面 1

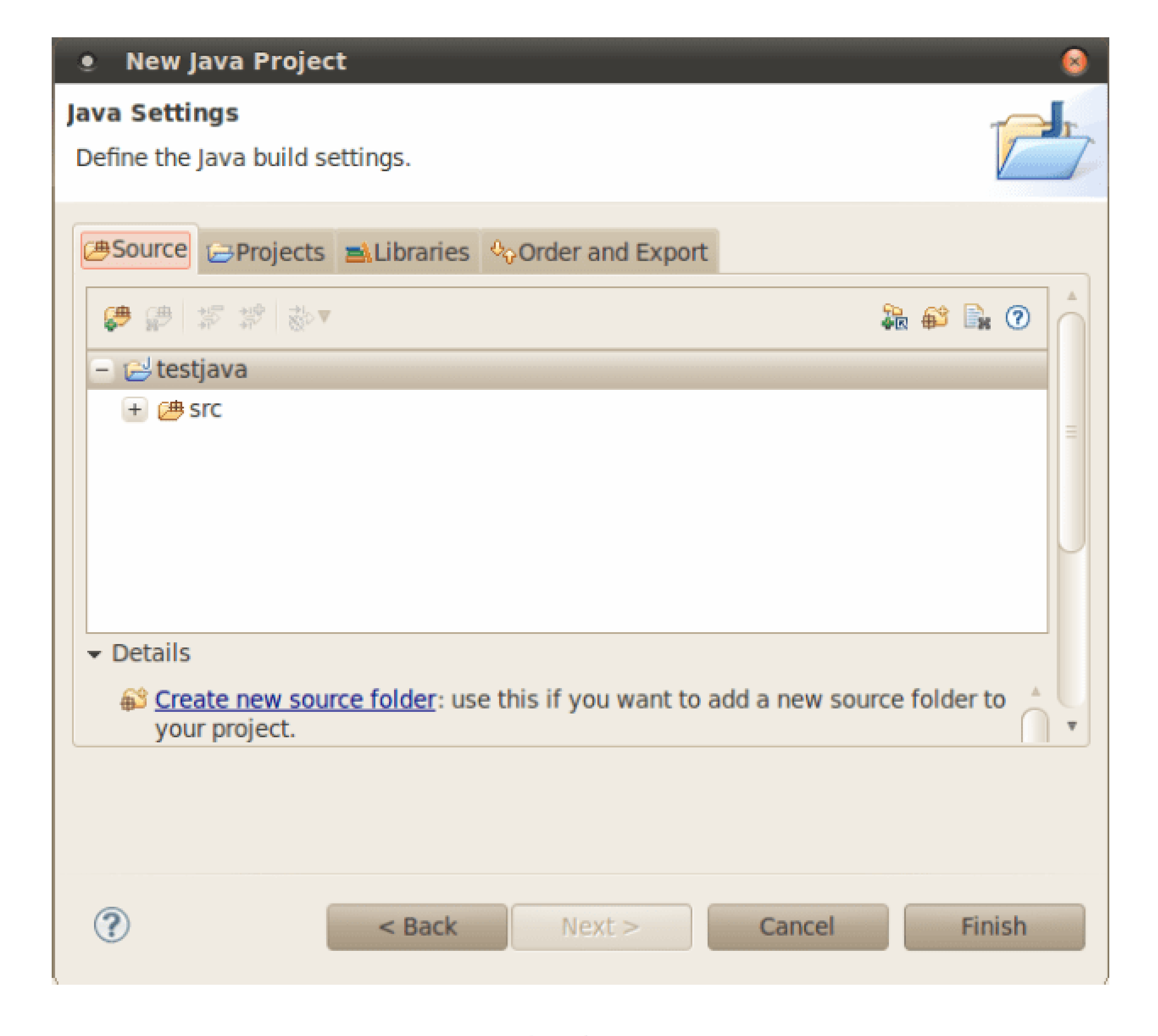


图 7.26　创建项目页面 2

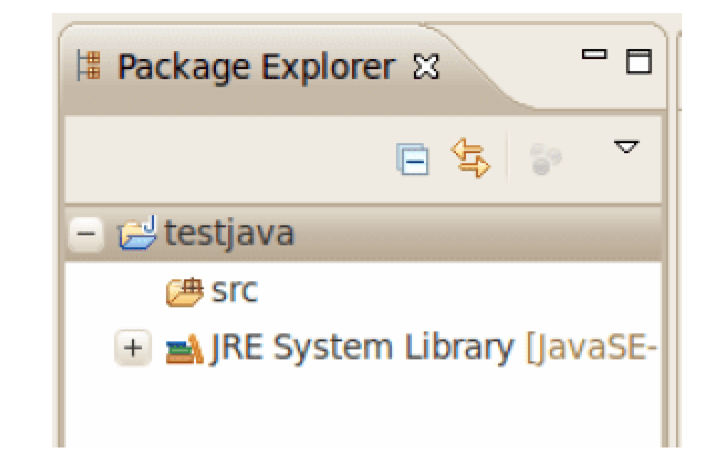


图 7.27　刚刚创建好的项目

之所以什么也没有，是因为我们什么也没创建。Java 项目，都是由一个一个的包，也就是常说的 Package 组成的，所以懒蜗牛同学现在需要创建一个 Package。于是他选择了 File|New|Package 菜单，Eclipse 弹出了创建 Package 的对话框，如图 7.28 所示。

懒蜗牛在 Name 文本框里面输入了包的名字：hello，并单击了 Finish 按钮。然后就可以看到懒蜗牛创建的 testjava 项目中，多了一个叫做 hello 的包，如图 7.29 这样。

【创建类】

好了，有了 Package 了，可是还没见到往哪写程序呀？懒蜗牛双击那个 Package 也没见有什么反应。这时候他才想起来：对了，这个面向对象的 Java，是离不开 Class，也就是类的。那就再创建个类吧。于是懒蜗牛又选择了 File|New|Class 菜单，Eclipse 弹出了如图

7.30 所示的创建类的窗口。

图 7.28　创建 package

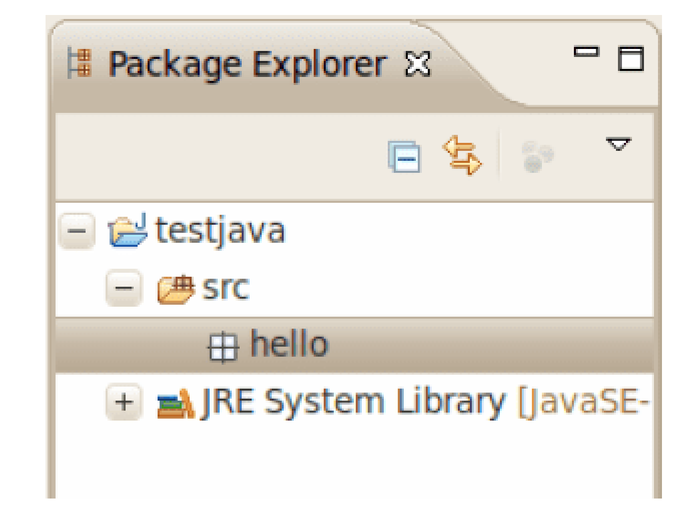


图 7.29　刚刚创建的 Package

图 7.30　创建类

在这个窗口中，懒蜗牛在 Name 文本框里填写了要创建的这个类的名称，叫做 HelloWorld。然后还勾选了下边那个“public static void main(String[] args)”复选框。意思就是，让 Eclipse 帮忙在这个类里创建好一个 main()方法——跟 C 语言一样，Java 也需要一个 main()方法，才知道程序从哪里开始执行。

选好了之后，懒蜗牛单击 Finish 按钮，于是又回到了 Workbench 界面，如图 7.31 所示。

【编写代码并运行】

好了，这回终于看见代码了，也就是那个 HelloWorld.java 文件的内容。懒蜗牛同学义无反顾地向 HelloWorld.java 文件中写进了最关键的一行代码，完成之后，HelloWorld.Java 中的全部代码如下：

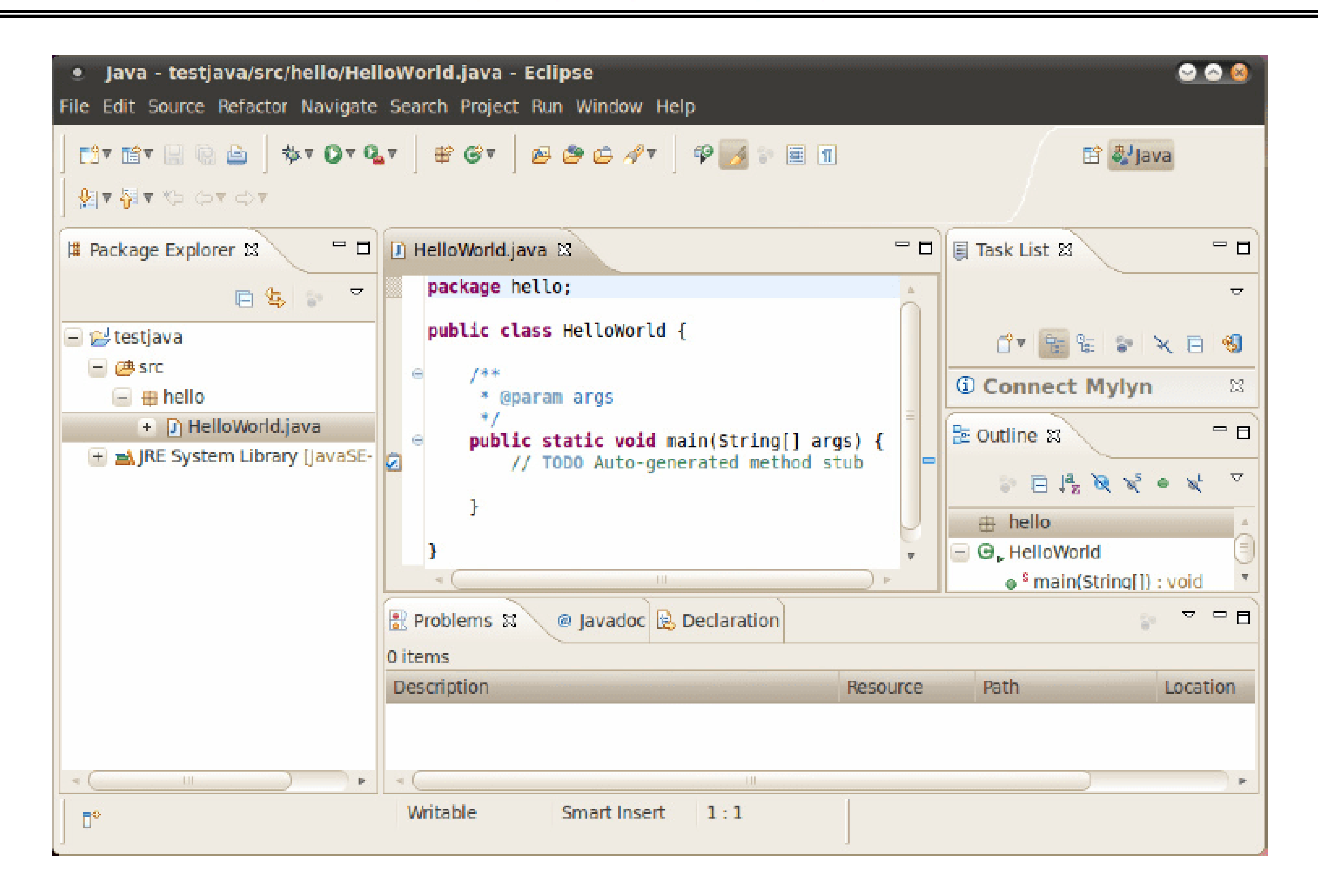


图 7.31　刚刚创建的类

```
package hello;

public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.print("Hello World Java!");
    }

}
```

好吧，其实懒蜗牛只是写了句 System.out.print("Hello World Java!")而已，这句代码的意思跟 C 语言里面的 printf()函数差不多，就是输出一个字符串。写好代码之后，单击 Workbanch 界面上边的那个绿色的运行按钮，就可以看到运行的结果了，如图 7.32 所示。

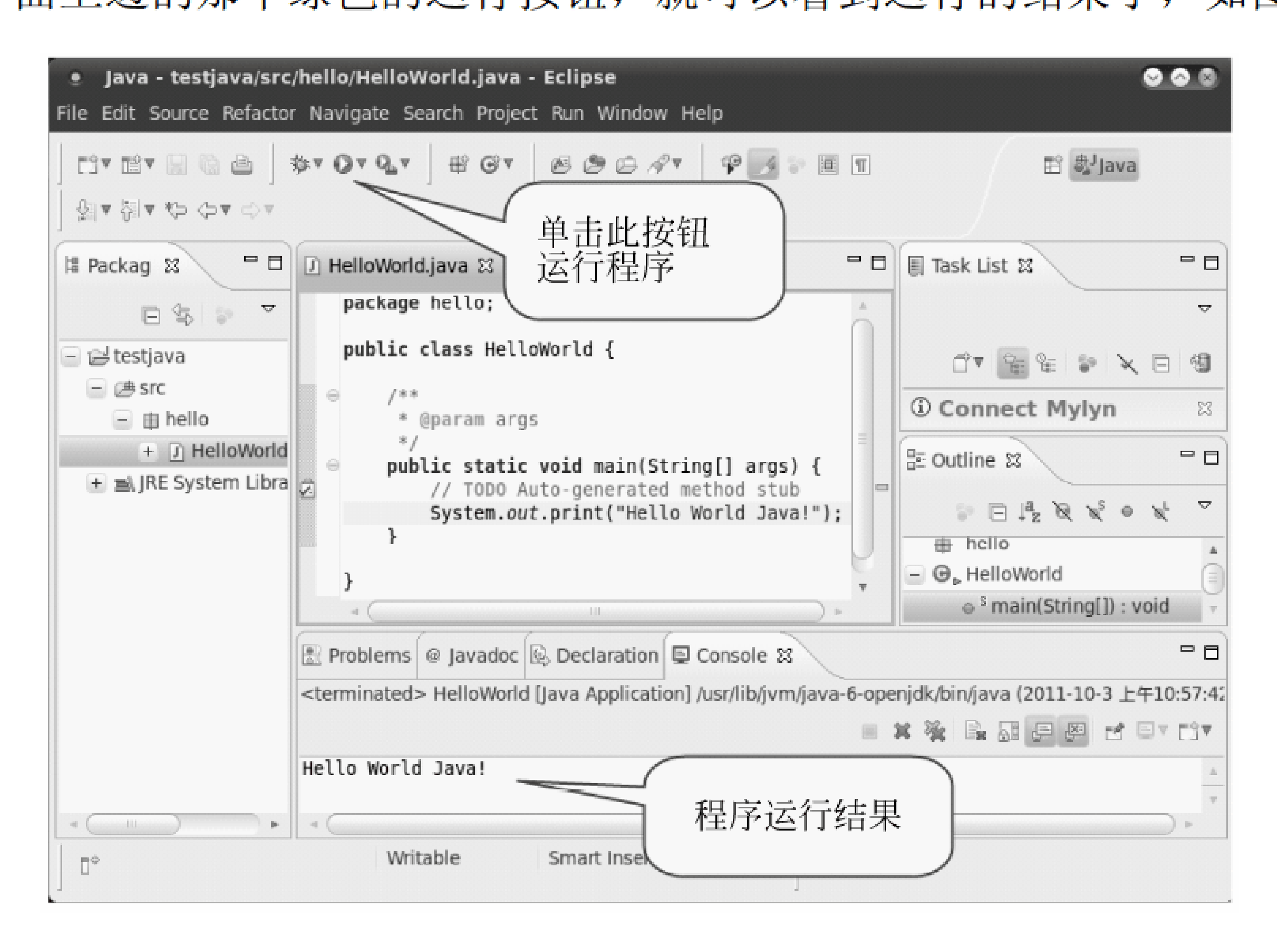


图 7.32　运行并查看结果

成功地写出了 Java 的 HelloWorld 程序之后，我们的懒蜗牛同学终于没有去尝试更多的语言。大概是从这 3 个最流行的编程语言中，就已经能够体会到各种语言编程的特点了吧。

7.4 Vim 编辑器的使用

前面咱们曾经介绍过几种常用的文本编辑器。其中使用最广泛，功能最强大的，就要数 Vim 和 Emacs 了（可别问哪个更强大啊，省得他俩又吵起来）。于是，懒蜗牛同学打算学习一下 Vim 的使用，为以后写程序及写脚本做好充分的准备（懒蜗牛学习 Vim 的事别让 Emacs 知道啊，低调低调）。

7.4.1 Vim 的操作模式

要学习 Vim，先要理解 Vim 的操作模式。Vim 不像 gedit、kate 之类的简单的文本编辑器，打开就能往里写字。如果你没有经过任何学习就在命令行中运行了 Vim，很可能急得胡敲乱打都输入不了一个字符，想退出都找不到门在哪。

懒蜗牛同学通过学习了解到，Vim 有 3 种工作模式，分别是：指令模式、输入模式、行末模式。

- 指令模式：Vim 运行起来之后，默认进入指令模式，如图 7.33 所示。在这种模式下，可以通过各种快捷键、组合键进行操作，对文本进行编辑（这个模式下，基本上满键盘都是快捷键，所以不要乱按哦）。例如可以执行复制、粘贴、删除、查找等操作。

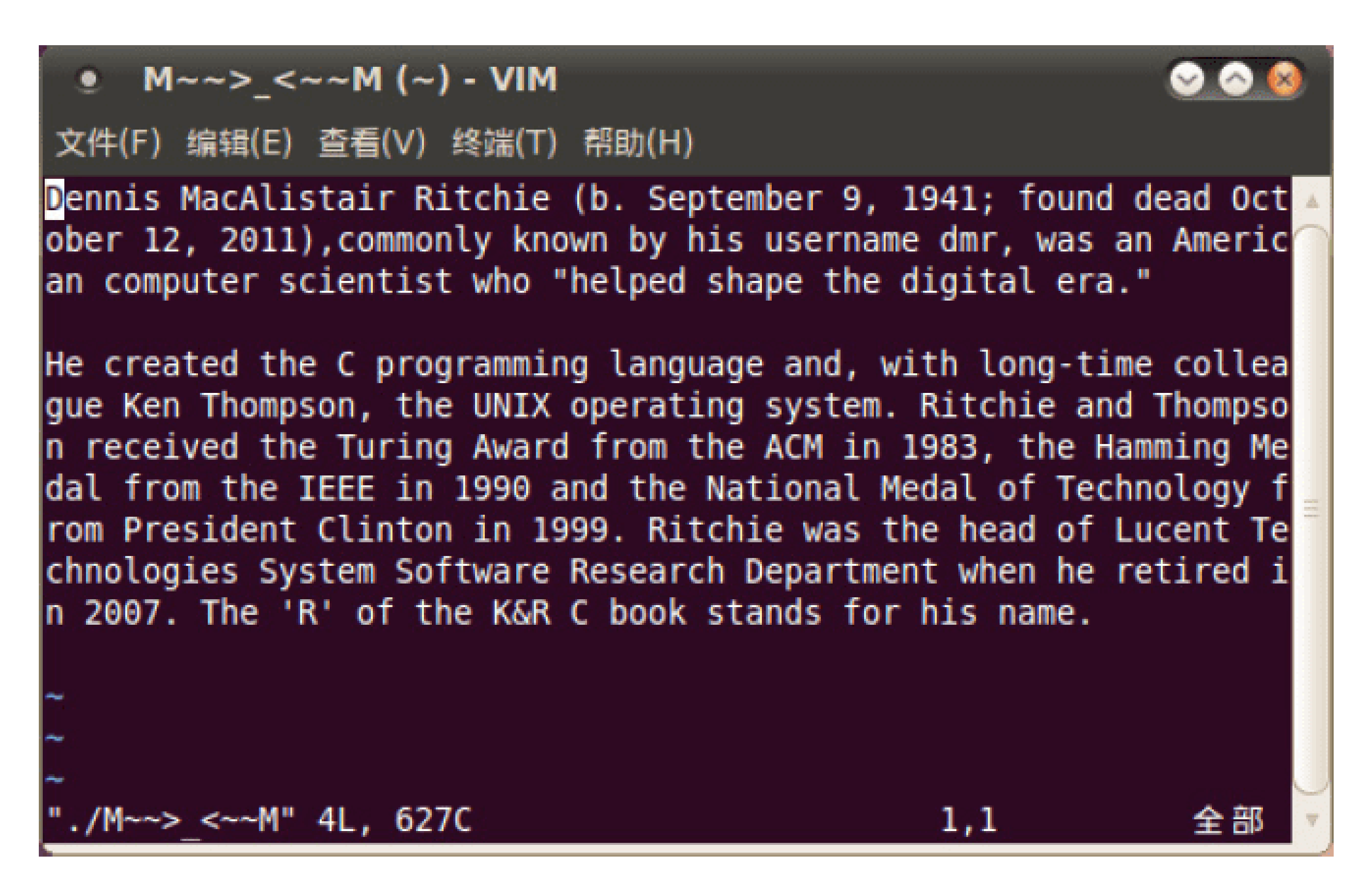


图 7.33 Vim 指令模式

- 输入模式：也就是普通的向文件里输入字符的模式。在指令模式下，按 i 键或者 Insert 键即可进入输入模式。进入输入模式后，界面左下角有“插入”字样，如图 7.34 所示。这时就跟使用 gedit 之类的编辑器一样了。在输入模式按 Esc 键可以回到指令模式。

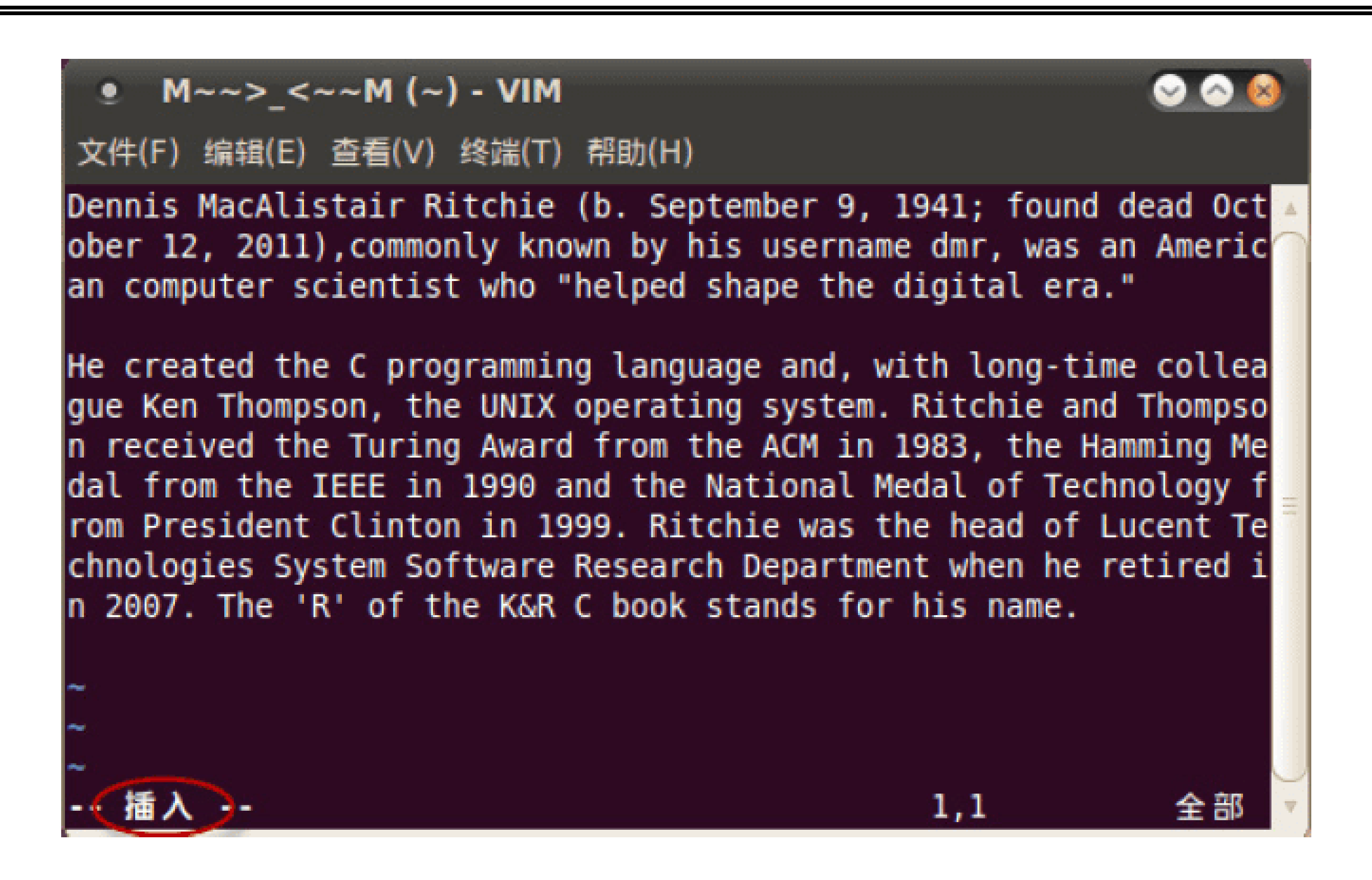


图 7.34　Vim 的输入模式

- 行末模式：这个模式下，可以在 Vim 界面的最下边一行输入命令（Vim 的命令），并且执行，来实现各种操作，如图 7.35 所示。例如可以执行打开文件、保存文件、查找替换、退出等等操作。在指令模式输入“:”，（也就是同时按下 Shift 和;键）即进入行末模式。此时光标出现在界面最后一行。行末模式执行完一条指令后自动回到指令模式（无论指令是否有效）。

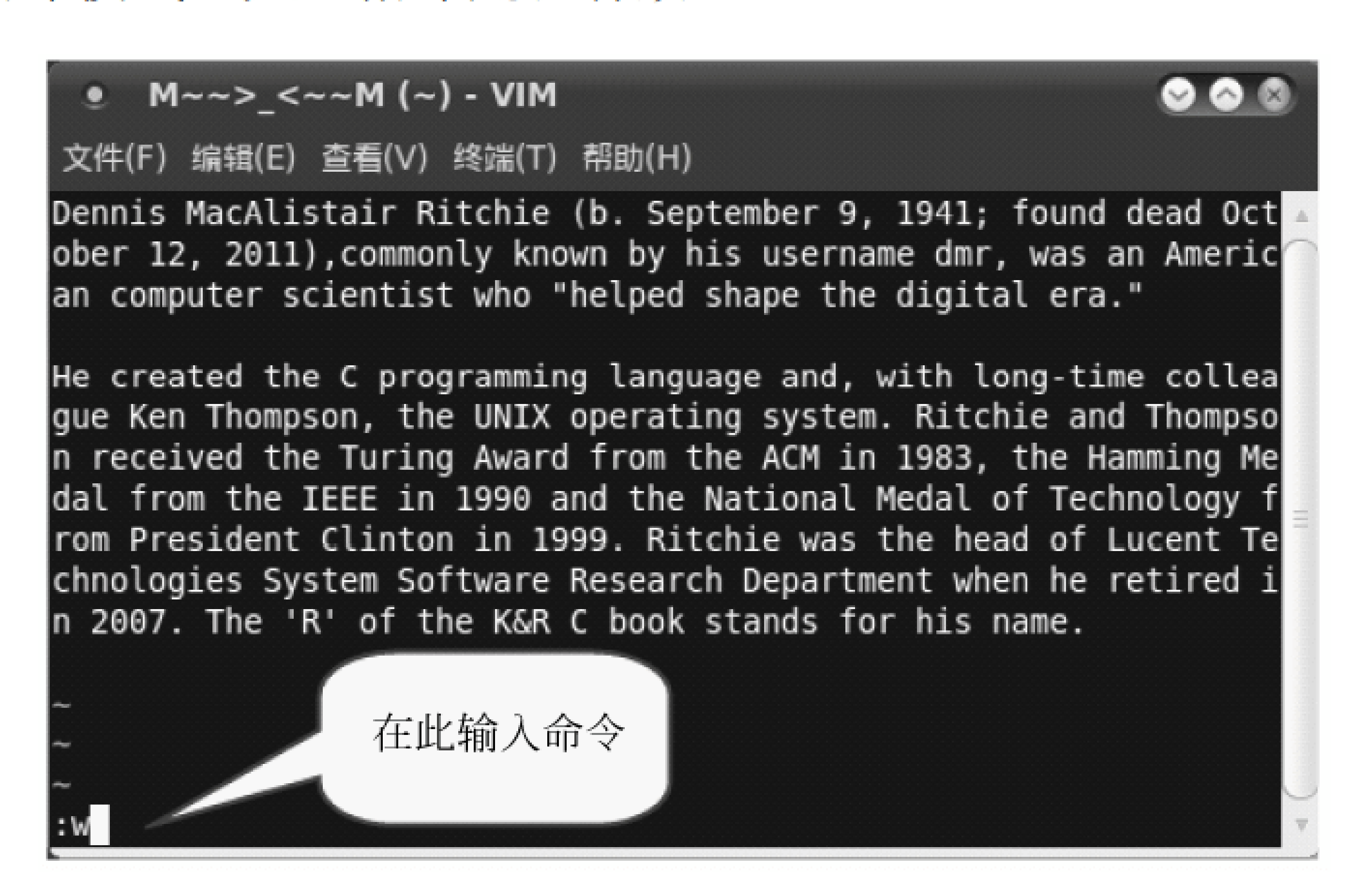


图 7.35　Vim 的行末模式

提示：为了简洁，也有时将 Vim 的操作归纳为两种模式——命令模式和输入模式。即将指令模式和行末模式统称为命令模式。

7.4.2　指令模式常用快捷键

了解了各种工作模式，懒蜗牛同学开始试着操作了。首先他打开了一个空的文件：

```
$vim ./test.txt
```

test.txt 文件是一个不存在的文件。运行这样的命令 Vim 并不会报错，而是会打开一个空白文件。等执行保存命令的时候，Vim 就会去创建这个文件。打开之后，懒蜗牛同学按 i 键进入了输入模式，并且输入了一些字符。输入了一大段之后，发现第 1 行有个字符写

错了，于是按向上的方向键，把光标移动到第 1 行（因为是在命令行中，无法使用鼠标直接定位），然后修改。修改之后，再按向下方向键，把光标移动到最后一行，然后继续写。刚写俩字母，发现第 1 行还是有错，于是再按向上方向键，把光标移动到第 1 行，修改完了再按向下的方向键，把光标移动到最后一行……

当懒蜗牛手指头抽筋以后，他终于意识到：应该有快捷键吧。

【定位相关快捷键】

是的，在指令模式下，有很多用于定位光标的快捷键，下面介绍几个常用的。

- Home、End、PageUp、PageDown 这 4 个键，依旧是它们原本的用途，依次是移动到行首、移动到行尾、上翻页、下翻页。这和在别的编辑器里面一样，就不多说了。
- Shift+g 组合键可以快速定位到指定的行。如果你想直接查看当前文件的第 168 行，那么就进入指令模式，直接依次在键盘上按下 1、6、8 这 3 个键（这时候输入的东西并不显示），然后按下 Shift+g 组合键，就可以看到光标直接跳到了 168 行的行首，效果如图 7.36 所示。另外，如果想移动到整个文件的最后一行，可以直接按两下 Shift+g 组合键（按住 Shift 敲两下 g）；如果想直接移动到文件的首行，可以直接按两下 g 键。

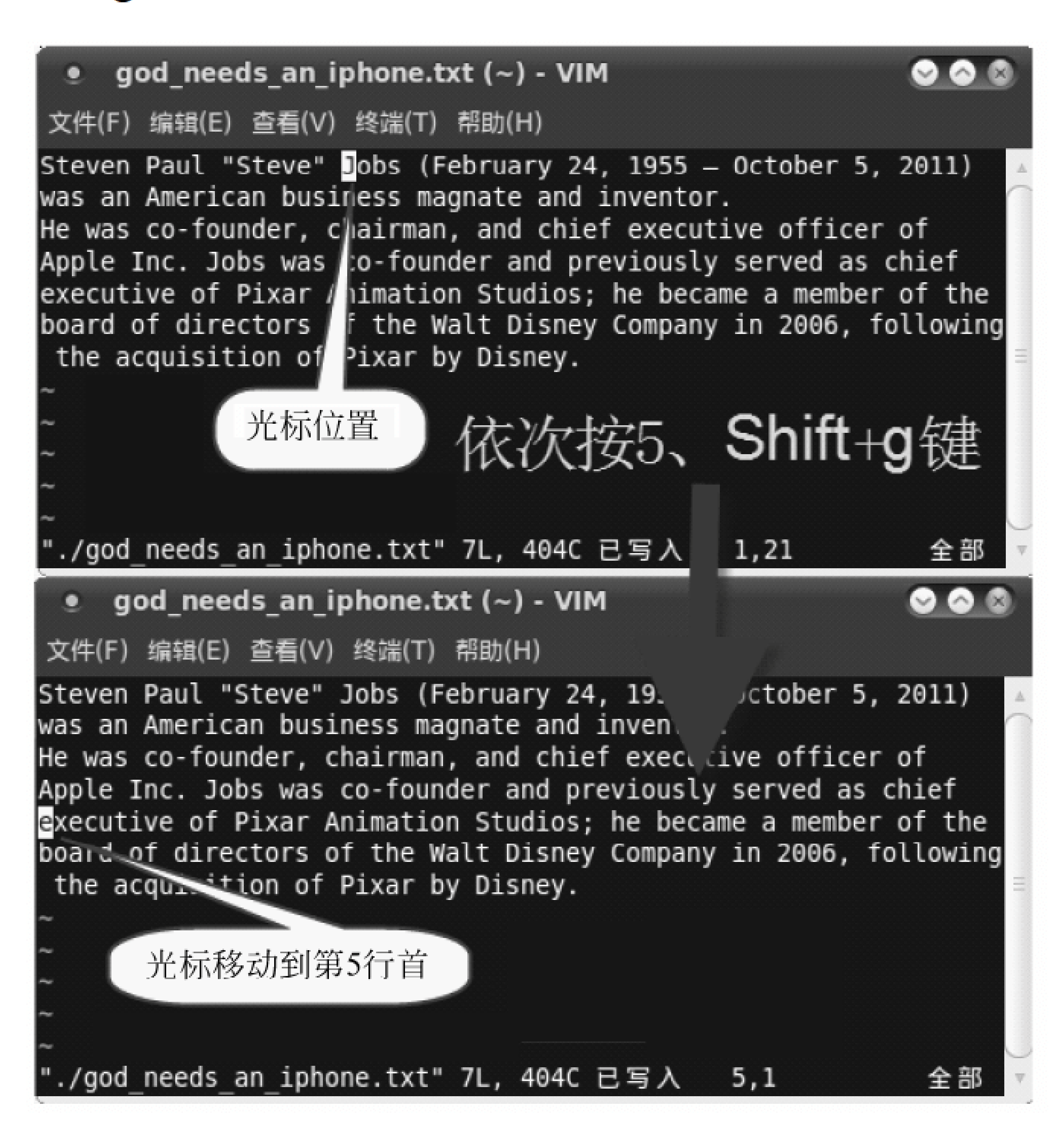


图 7.36　跳转到指定行

- w 和 b 键，分别表示向后和向前移动一个单词。这在看英文的时候比较有用。移动一个单词后，光标将停在该单词的第一个字母。图 7.37 中描述的就是按 w 键的效果示意。

【编辑相关快捷键】

文档编辑器嘛，自然少不了对文档的编辑。在编辑的过程中，也有很多快捷键可以帮

忙提高效率。

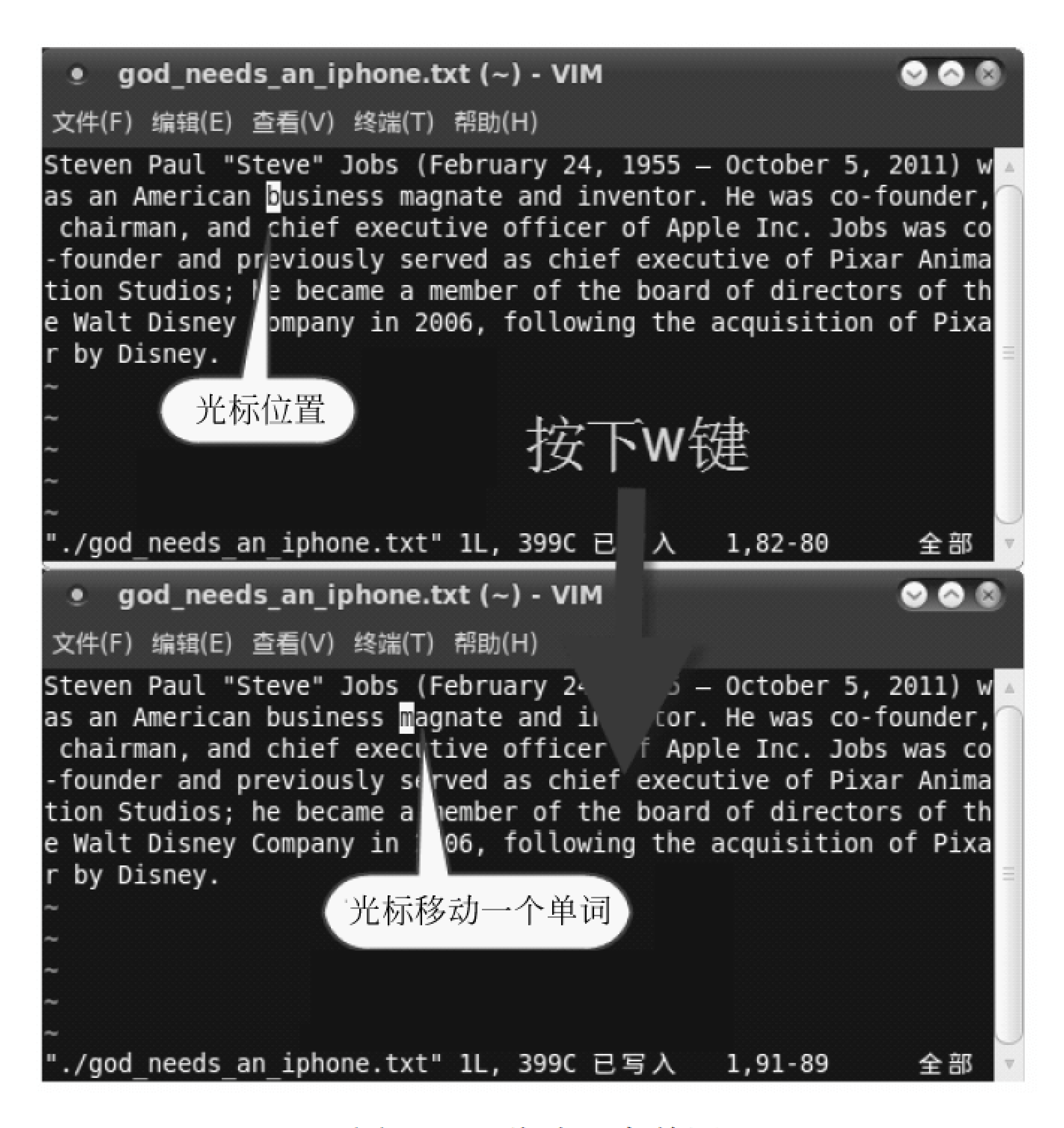


图 7.37 移动一个单词

- 使用 d 键来删除指定数量的行。例如，想删除从当前光标所在行向下数 3 行的内容（包括当前行），则可以依次按下 d、3、d 这 3 个键，如图 7.38 所示；如果想删除当前行向下 14 行的内容，则可以依次按 d、1、4、d 这 4 个键；如果想删除当前行，则可以直接连续按两下 d 键。

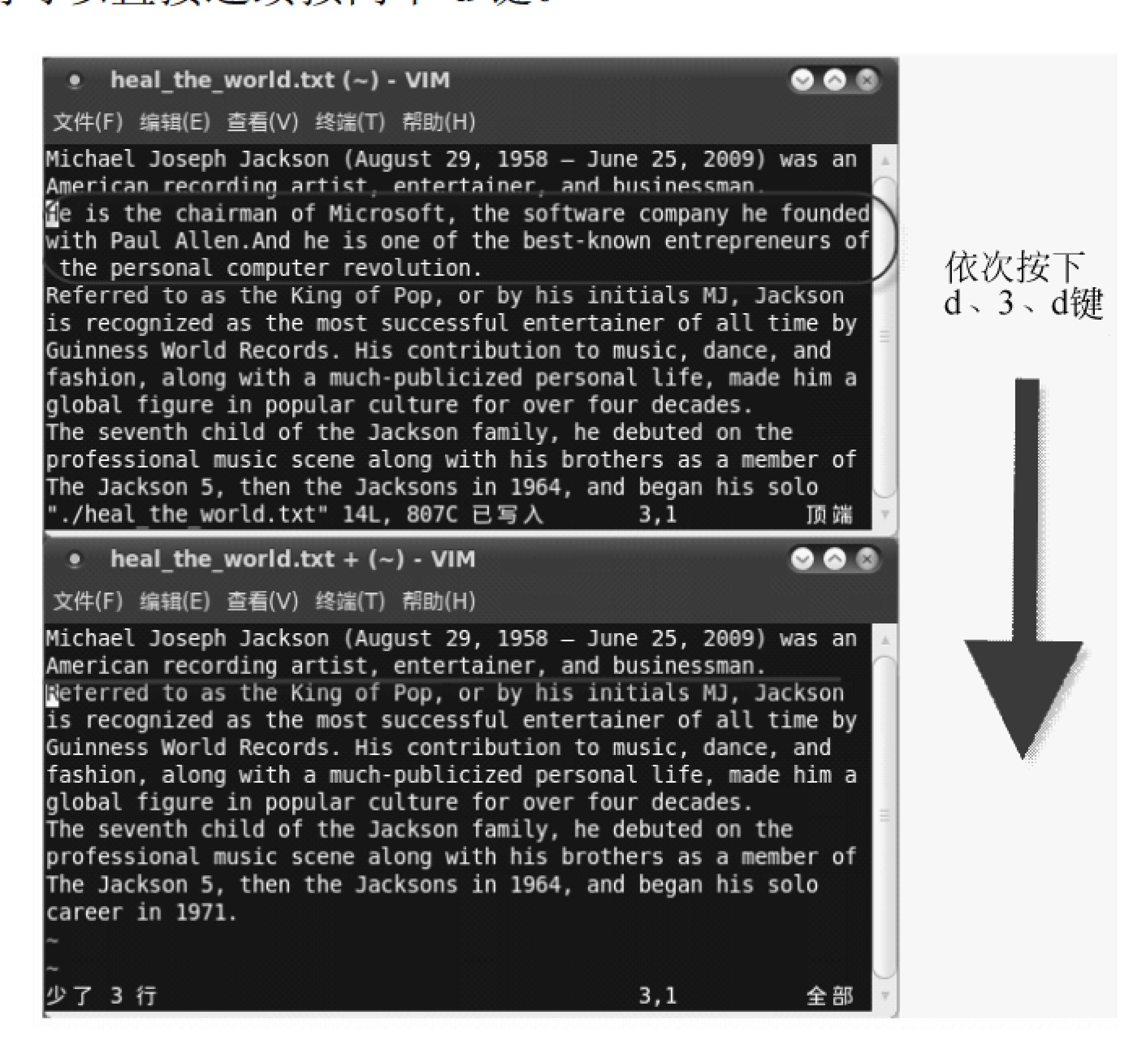


图 7.38 删除指定数量行

- 同样可以使用 d 键来删除指定数量的字符。依次按下 d、3、向右方向键，则删除从当前字符算起，向右 3 个字符（包括当前字符）；依次按下 d、1、4、向左方向键，则删除当前字符左侧的 14 个字符（不包括当前字符）。
- Shift+j 组合键，用于删除当前行末尾的回车。换句话说就是可以将下一行并到当前行的末尾，如图 7.39 所示。

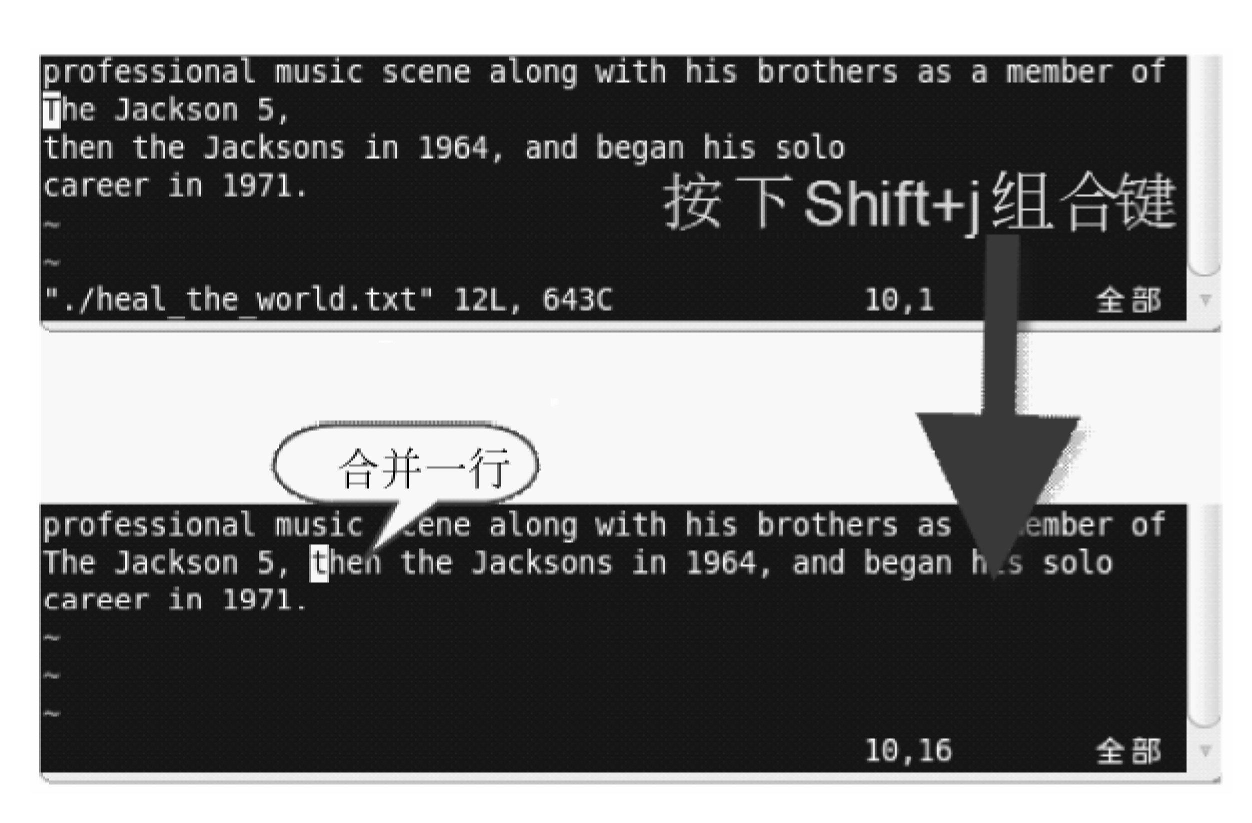


图 7.39 删除行末的回车

- 编辑的时候免不了出错，如果有误操作，可以按 u 键撤销上一次的操作。
- 如果撤销错了，可以按 Ctrl+u 组合键来恢复上一次撤销的操作。
- 最常用的“吭哧 C，吭哧 V”大法在 Vim 中更是出神入化。使用 y 键可以复制指定的行数和字符数，用法跟 d 键类似。例如按下 y、3、y，则复制当前行向下 3 行的内容（包括当前行）；而 y、2、向左方向键，表示复制当前字符左侧的 2 个字符（不包括当前行）。
- 复制内容之后，按 p 键粘贴。如果复制的是字符，则会粘贴到当前光标所在位置，光标处原有内容顺次右移；如果复制的是整行，则会粘贴到当前光标所在行的下一行的位置，原有的行顺次向下移动。图 7.40 是整个复制粘贴过程的示意。



图 7.40 复制粘贴

【查找相关快捷键】

查找内容，也是一个编辑器的基本能力。在 Vim 中，查找指定的字符串有以下几种快捷键。

- ❑ 在指令模式中按下/键，会看到 Vim 界面左下角出现“/”符号，然后输入想要查找的字符串并回车，Vim 将从当前光标位置开始向下查找。如果找到，光标将跳转到第一个搜索结果的位置，如图 7.41 所示。

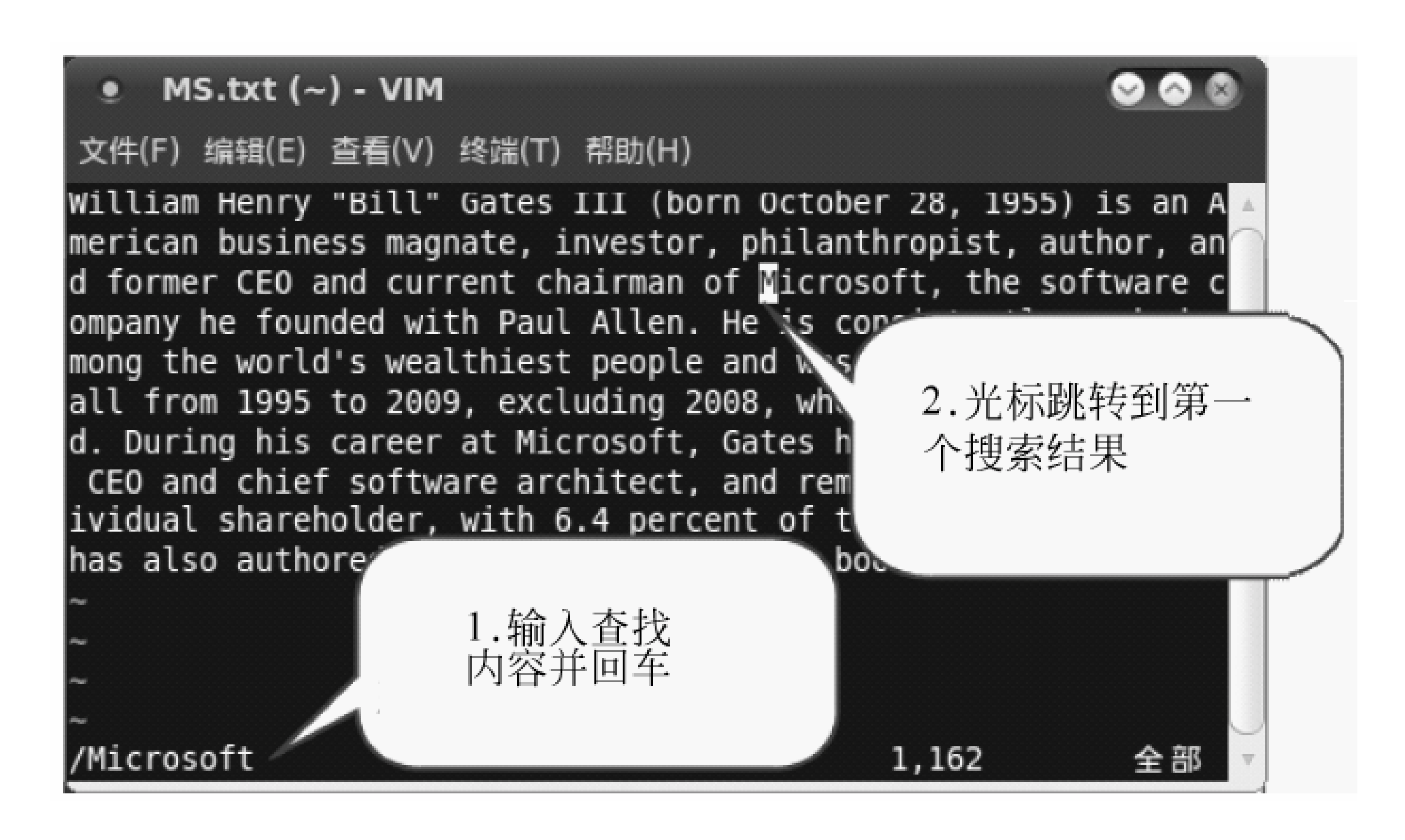


图 7.41　查找字符串

- ❑ 在指令模式中按下 Shift+/组合键，会看到 Vim 界面左下角出现“?”符号，然后输入想要查找的字符串并回车，Vim 将从当前光标位置开始向上查找。如果找到，光标将跳转到第一个搜索结果的位置，如图 7.42 所示。



图 7.42　反向查找字符串

- ❑ 将光标移动到要查找的某个单词的位置，然后按 Shift+8 组合键，则 Vim 从当前位置向下查找该单词；或者按 Shift+3 组合键，则 Vim 从当前位置向上查找该单词。找到后，光标自动跳转到第一个搜索结果的位置。图 7.43 描述的就是 Shift+8 组合键的使用。
- ❑ 无论使用哪种查找方式，在找到一个结果后，按下 n 键，则继续向同方向查找下一个结果；按下 Shift+n 组合键，则向反方向查找上一个结果。如果找到，则光标自动跳转到该搜索结果的位置。

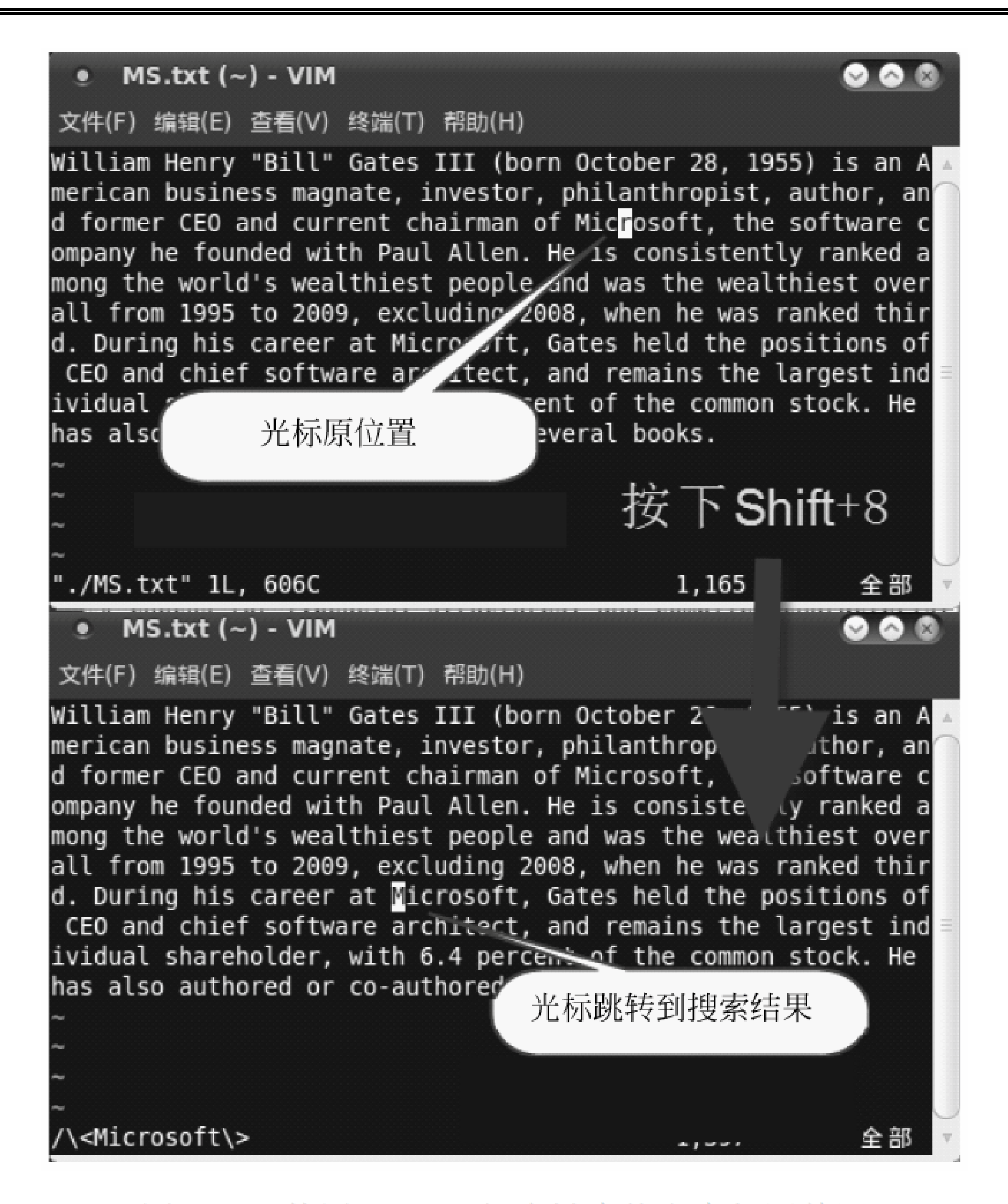


图 7.43　使用 Shift+8 组合键查找文中相同单词

7.4.3　行末模式常用命令

看到这里，您已经比懒蜗牛同学更加了解 Vim 了。因为他还没有记住那么多的快捷键，只是稍稍体验了一下，写了几行文字，就准备保存并退出了。

要想保存文档，需要进入行末模式。于是懒蜗牛按下了 Esc 键，进入了指令模式，然后又按下 Shift+;组合键，就看到左下角出现了一个“:”，这说明已经进入行末模式了，如图 7.44 所示。

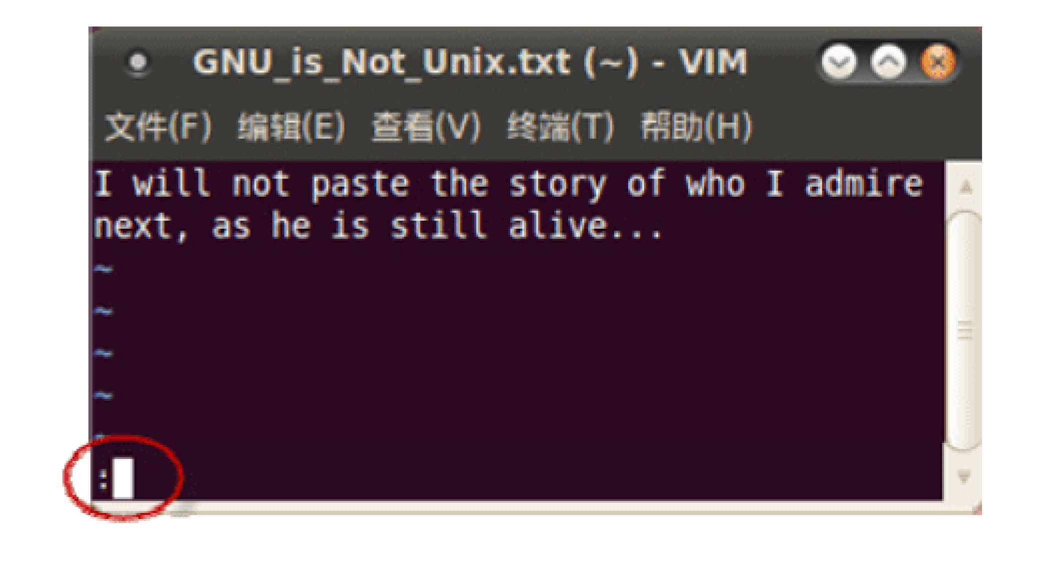


图 7.44　行末模式

在这个“:”的后面，可以输入很多复杂的命令，实现各种苛刻的操作要求。咱们分门别类地说说吧。

【文件操作命令】

跟文件操作相关的命令是最常用的了，懒蜗牛同学要保存他的文件就需要用到这类命令。

- “w”命令——这个命令用于保存当前文件。也可以在 w 后面加上文件名，例如“:w back1.txt”，则代表将当前文件另存为新的文件名“back1.txt”。
- “new 文件名”——这个命令用于打开或者新建一个文件（同时关闭当前文件）。如果“文件名”指定的文件存在则打开，否则新建（但是要调用 w 命令的时候才

真的存储该文件）。

- “q”——退出 Vim 程序。如果打开的文件没有保存，会提示错误，无法退出。此时如果确认要放弃修改，强制退出，则可使用“q!”命令。

懒蜗牛同学目前就是需要保存文件，并且退出。他可以依次输入 w 和 q 命令，不过他更加灵活地运用了 Vim 的特性，直接在“:”后面输入了“wq”并回车，于是 Vim 就先保存好了懒蜗牛同学修改的文本，然后退出了。

【艰巨的任务】

这一日，懒蜗牛同学又收到了 MM 同志的一项请求：有一个很大的文本文件，是一个聊天软件的聊天记录。每一行对话开头都有一个几点几分的标识，记录着对话的具体时间。MM 想把这些时间删掉。懒蜗牛同学一想，这个要求听起来复杂，不过应该难不倒 Vim 吧，于是就答应了。

拿到 MM 发来的文件打开一看，里面大约是这么个形式（为了不泄露隐私，咱们就用“甲”和“乙”来代替聊天人的名字了）：

```
20:28 甲:那什么小王，要不你别来了啊
20:29 乙:怎么了?
20:30 甲:现在都 8:30 了
20:31 乙:嗐……

2010 年 1 月 17 日
17:34 甲:您辛苦!
17:35 乙:嗯
17:38 甲:昨天我到您家了
17:40 乙:啊!
17:41 甲:啪啪啪这么一打门啊
17:43 乙:这……
17:43 甲:里边出来一人
17:44 乙:是
17:45 甲:我一看不是外人
17:47 乙:去你的吧
17:50 甲:……
```

以上只是其中一小段，总共的行数不少，足有 3000 多行。形式就是上面这样，以日期分隔的很多段聊天记录，每句话前都有具体的时间。MM 的要求是删掉这些具体的时间信息，这样看着清爽一些。但是像“2010 年 1 月 17 日”这样的日期信息不要删除，方便查找。还有对话里出现的时间，当然也不要删除。

如果用手一个一个删，估计懒蜗牛同学的手指头又要抽筋了。不过好在他现在已经不是菜鸟级别的使用者了，而是已经会用无比强大的 Vim 的入门用户啦！

【查找替换】

MM 交代的这个工作，就需要使用 Vim 的行末模式中的查找替换命令了。这个命令大致的格式是这样：

```
:查找范围 s/查找内容/替换内容/
```

下面仔细说说每一项。

- 查找范围——就是在哪里查找。这个范围可以是整个文件，也可以是文件中指定的几行或一行。如果要在当前行查找，那么就不写查找范围；如果要在整个文件

中查找，那么就写一个“%”符号，表示全文查找；也可以写用逗号分隔的两个数字，例如“3,19”，意思就是从第 3 行开始找，找到第 19 行为止（包括第 3 和第 19 行）。

提示：可以使用“$”符号代表文件最后一行，“6,$”表示从第 6 行开始，一直查找到最后一行。

- 查找内容——也就是想要查找的内容，这里可以使用正则表达式。
- 替换内容——需要把查找到的内容替换成的字符串。如果为空，则表示删除查找到的内容。

举个例子，比如我们想要在文件的第 3 行到第 50 行中，查找所有的 newbie，替换成 expert。那么就在行末模式中输入如下命令：

```
:3,50s/newbie/expert/g
```

这样一个命令，就让所有的 newbie 都变成 expert 了。这条命令的最后一个“g”参数要说明一下。这个参数意味着替换掉每行所有的 newbie，如果不加这个参数，那么当某一行找到了一个 newbie 并替换以后，就不再查找这一行，直接继续找下一行去了。

那么懒蜗牛同学的任务应该如何完成呢？用这个查找替换的命令，加上正则表达式就可以实现了。只见懒蜗牛同学运行了如下命令：

```
:%s/^\d\d:\d\d//
```

稍微解释一下这个命令吧，其实都是介绍过的东西。

- “%”符号代表要在整个文件中查找。
- 要查找的内容是“^\d\d:\d\d”，其中“^”代表一行的开头，因为我们只需要找每行开头的时间，对话中的时间不需要匹配；“\d”咱们说过，匹配一个 0～9 的数字。那么“^\d\d:\d\d”用人类语言描述出来就是：每行开头出现的、由冒号分隔的、前后各有两位数字的这样的字符串。
- 要替换的内容是空，意味着删除查找到的内容。
- 最后没有加“g”参数，意味着我们只替换每行找到的第一个（其实我们明确写明了找行首的字符串，所以肯定只能找到一个。因为每行只可能有一个行首）。

于是，懒蜗牛同学只使用了这样一个命令，就解决了 MM 的复杂要求。

【过滤内容】

然而 MM 的要求并不满足于此，堪称“十万个怎么办”的 MM 同志又给懒蜗牛提出了更高的要求。这回，还是那段聊天记录，经过懒蜗牛的处理后，已经是这样了：

```
甲:那什么小王，要不你别来了啊
乙:怎么了？
甲:现在都 8:30 了
乙:嗐……

2010 年 1 月 17 日
甲:您辛苦！
乙:嗯
甲:昨天我到您家了
乙:啊！
```

```
甲:啪啪啪这么一打门啊
乙:这……
甲:里边出来一人
乙:是
甲:我一看不是外人
乙:去你的吧
甲:……
```

MM 同志觉得这聊天记录充满喜感，简直可以改编成相声了。于是 MM 让懒蜗牛把这个文件里面，甲和乙的话分别存成两个文件，当作台词发给两位同事，以便他们在公司年会上表演相声。也就是说，MM 希望把这一个文件拆分成两个文件，比如叫“甲.txt”和“乙.txt”。那么这两个文件的内容大约就是这样：

甲.txt 文件：

```
甲:那什么小王，要不你别来了啊
甲:现在都 8:30 了

2010 年 1 月 17 日
甲:您辛苦!
甲:昨天我到您家了
甲:啪啪啪这么一打门啊
甲:里边出来一人
甲:我一看不是外人
甲:……
```

乙.txt 文件：

```
乙:怎么了?
乙:嗐……

2010 年 1 月 17 日
乙:嗯
乙:啊
乙:这……
乙:是
乙:去你的吧
```

这个要求如何满足呢？也不难，需要用到内容过滤的命令。我们可以用行末模式中的“g”命令来过滤出包含某字符串的行，也可以用“v”命令来过滤出不包含某字符串的行。并且可以在过滤之后对找到的行进行进一步处理。命令格式大约是这样：

```
:g/要过滤的字符串/对过滤出的内容的进一步操作
```

比如 MM 这次的要求，就可以总结为如下两个操作。

（1）删除所有包含“甲:”字符串的行，并另存为“乙.txt”。

（2）删除所有包含“乙:”字符串的行，并另存为“甲.txt”。

因此就可以使用 g 命令，例如这样操作：

```
:g/甲:/d
```

当然，为了保险，我们还可以更严格地限定只过滤每一行开头为“甲:”字符串的行：

```
:g/^甲:/d
```

后面的“d”的意思就是删除行。也就是对所有由 g 命令根据字符串“甲:”过滤出来的行，执行删除操作。这之后，再运行命令：

```
:w 乙.txt
```

就将文件另存为了“乙.txt”文件。另一个“甲.txt”也如法炮制即可。

当然，我们也可以用 v 命令，过滤出所有不带“甲:”的行，就是这样：

```
:v/^甲:/w 乙.txt
```

这个命令的意思就是过滤出所有不带“甲:”的行，然后将过滤内容直接存为“乙.txt”文件。

虽然 MM 的要求总是很复杂，不过懒蜗牛倒是都一一满足了。但这不过是 Vim 编辑器的牛刀小试，他的文本编辑功能，基本上已经强大到只有你想不到，没有他做不到的程度了。

7.5 本章小结

这些日子，咱们这位懒蜗牛同学可是没少忙活。又是 C，又是 C++，又是 PHP，又是 Java 的，配置了不少语言的开发环境。并且为了更好地编程，懒蜗牛还学习了 Vim 的使用。可是也没见他写出什么有用的东西来。当然，也许生活，就是一个 HelloWorld，接着另一个 HelloWorld 吧。

等 HelloWorld 写腻了，也许他就该用 Vim 去写点有意思的程序了。

第 8 章　程序是怎样炼成的

懒蜗牛同学搭建好了各种语言的开发环境，并且开发出了不少的 HelloWorld。但他显然并不想就此罢休。凡事都喜欢刨根问底的懒蜗牛同学，还要搞明白一个软件从源代码到打包为成品的整个过程。

8.1　施　工　队

软件的最初状态自然就是源代码，要把 C 语言的源代码变成二进制的程序，离不开 GCC。那么 GCC 到底对源代码做了什么，才能把一段段冰冷的代码变成一只只鲜活的程序呢？（为啥要用“只”……）

8.1.1　懒蜗牛的日记 A

“2010 年 9 月 3 日　降雨

学了这么几天，觉得还是 C 语言最有意思、最灵活高效。听说好多 Linux 系统的软件都是拿 C 语言写的，连 Linux 内核也是以 C 语言为主创造的。我能不能也写出一个自己的小软件来呢？不要求功能有多么复杂，但要符合 Linux 的精神——只做一件事，但要做到最好。

仅仅编译一个 HelloWorld 很简单，只有一个.c 文件，只用一条命令。我应该设计些复杂的程序，实现点有意思的功能才会有进步。编个什么程序好呢？”

8.1.2　编译多个源文件的程序

懒蜗牛同学之前写过一个 HelloWorld 程序，但是觉得那个程序实在太简单，没什么意思。于是这回他决定写个稍微复杂点的程序。

【无聊的 rubbish 1 号】

只见他冥思苦想之后，写下一个源码文件，叫做 rubbish.c，看来这懒蜗牛同学还真谦虚。在 rubbish.c 文件中写了些代码后，懒蜗牛就叫来 GCC 进行编译，运行了这么个命令：

```
$gcc ./rubbish.c -o rubbish
```

命令后面那个“-o rubbish”的意思，就是指定编译后的二进制文件的文件名叫做 rubbish。这样就不会每回都输出为 a.out 文件了。编译出来后，还冒着热气的 rubbish 就被懒蜗牛叫进内存运行起来。只见 rubbish 飞身跳进内存，跑进内存后抢过标准输出设备，

对着那个设备大喊一声："I am a Rubbish!"然后，就跑回硬盘继续睡觉去了。我说懒蜗牛同学呀，不是说设计个复杂点的程序吗？您这个 rubbish 跟那个 HelloWorld 有啥区别呀？

提示：标准输出设备，即/dev/stdout 设备文件，一般该文件映射到当前字符终端。

我们后来管这个弱智的程序叫做 rubbish 1 号，因为懒蜗牛同学很快又创造出了很多的 rubbish。

【同样无聊的 rubbish 2 号】

不一会儿，懒蜗牛又拿来 rubbish 1 号的图纸改起来。10 分钟后，图纸完成，交给 GCC 编译，懒蜗牛很自觉地把这个程序命名为 rubbish2。

```
$gcc ./rubbish.c -o rubbish2
```

很快，rubbish 2 号诞生！毫无悬念地，懒蜗牛马上让我叫醒 rubbish 2 号起来干活。于是我走进硬盘，叫醒 rubbish 2 号。只见 rubbish 2 号立刻飞身跳进内存，依旧是对着标准输出设备大喊一声："I am a Ru~Ru~Ru~Rubbish~~~~!"喊完了就跑回去继续睡觉了。懒蜗牛同学成功地利用 for 循环创造了一个结巴。

【多个文件的 rubbish 3 号】

15 分钟后，rubbish 3 号的图纸再次毫无悬念地完成。这回的图纸不光是一个 rubbish.c 文件了，而是包含了 3 个文件：rubbish.c、input.c 和 input.h。这回懒蜗牛依旧是叫来了 GCC，他运行这么个命令：

```
$gcc rubbish.c input.c -o rubbish3
```

这就是当代码包含多个.c 文件时候的编译方式。有人问，那个.h 文件呢？.h 文件一般是要被包含进某个.c 文件的，这个包含的动作一般在预处理的时候就给处理了，不需要在编译命令里写上.h 文件。至于什么叫预处理，您别忙，咱们一会儿就会说到。

rubbish 3 号这会儿已经起床，跑进内存向懒蜗牛发问："How many Ru do you want？"然后就等待懒蜗牛输入。懒蜗牛同学输入了一个 6，于是就听见 rubbish 3 号大喊："I am a Ru~Ru~Ru~Ru~ Ru~ Ru~Rubbish!"——程控结巴！

【系统调用的 rubbish 4 号】

rubbish 4 号的图纸诞生啦。Vim 告诉我说，这回懒蜗牛调用了创建线程的库函数（因为懒蜗牛是用 Vim 编程的嘛，所以 Vim 能知道写了些什么）。果然，在编译的时候懒蜗牛运行了：

```
$gcc rubbish.c input.c -lpthread -o rubbish4
```

其中，-lpthread 的意思就是要连接创建线程相关的库函数。图纸交给 GCC 后，很快 rubbish 4 号诞生，并且很快就起床跑进内存。只见他念动咒语"唵木哒咪咪呀……分！"然后白光一闪，rubbish 4 号变成了 2 个！2 个 4 号同时喊：

"I am a Ru~Ru~Ru~Ru~Rubbish!"

"I am a Ru~Ru~Ru~Ru~Rubbish!"

二重结巴！

8.1.3　编译过程详解

刚才咱们见到了编译一个（或一坨）C 语言源文件时的操作，下面就仔细说说编译的过程。

在编译的时候，虽然用户用的只有一个“gcc”命令，然而我们说过，GCC 是一个编译器套装。他不是一个人，不是只有一个“gcc”命令就能完成整个编译过程的！他们是一个团队，一个以“gcc”命令为首的源代码编译施工队。

施工队主要成员有 gcc、cpp、as 和 ld 这么 4 个命令。其中 gcc 命令是老大，其他几个干什么活都得听他的调遣，用户一般也只跟 gcc 命令打交道。当写好了图纸之后（也就是源代码，就比如刚才懒蜗牛写的 rubbish.c），虽然用户直接把图纸交给 gcc 命令就不管了，但其实 gcc 命令需要去调动其他命令进行各种处理，才能完成编译工作。

（以下文中出现的小写“gcc”均代表 gcc 命令，而不是整个 GCC 开发套件。）

【施工第 1 步——预处理】

一般来说，gcc 拿到图纸后，会首先叫来 cpp 进行预处理。

预处理主要就是将文件里的宏定义进行展开。什么是宏定义呢？人类用户一般都比较懒，或者说，人类能力有限，不愿意写很多重复的，相似的东西，就把这些都定义成宏。比如，这么写：

```
#define TOTAL_NUMBER 18353226
```

这就是定义一个叫做 TOTAL_NUMBER 的宏，从名字看，这个宏代表了一个总数，数值是 18353226。那么以后再要用到这个总数的时候，就直接写 TOTAL_NUMBER 就可以了，不用写那一大串数字。而且，如果总数变了，只要在最初#define 的位置修改一次就可以，反正就是为了偷懒。

cpp 的任务就是把这类的宏定义都替换回去。把所有的 TOTAL_NUMBER 都替换成 18353226；把所有#include 引用的文件内容都粘贴进来等。这么说可能不形象，那咱们找个实例吧，比如有一个 hello.c 文件，内容是这样的：

```
#include <stdio.h>
#define TOTAL_NUMBER  18353226

int main(int argc, char* argv[])
{
    int i;
    for (i=0;i< TOTAL_NUMBER;i++)
    {
        printf("Hello World!\n");
    }
    return 0;
}
```

那么经过了预处理之后的内容是什么样呢？如果您想体验一下，可以自己试着运行：

```
$gcc -E ./hello.c > hello_cpp.c
```

“-E”参数的意思就是让 gcc 只对 main.c 进行预处理。这样处理后的结果会直接输出到标准输出，不方便查看。所以我们需要进行输出转向，把输出的内容存进 hello_cpp.c 文

件里。打开这个文件你就会发现，仅仅几行的程序，预处理后变成了 800 多行！

提示：gcc 是调用 cpp 命令来进行预处理，因此也可直接用 cpp 命令进行预处理，运行以下命令：

```
$cpp ./hello.c > hello_cpp.c
```

效果与“gcc –E”相同。

这 800 多行就是因为这段代码包含了 stdio.h 文件，stdio.h 文件又包含了很多其他的.h 文件，于是在预处理的时候，cpp 就把这些.h 文件全部粘贴了进来（但其实都没什么用，我们只用到了其中的 printf()函数）。

预编译后的文件大约就是下面这个样子：

```
# 1 "./hello.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "./hello.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 916 "/usr/include/stdio.h" 3 4
/*为防止出版社说我骗稿费，此处省略 800 多行……*/
# 2 "./hello.c" 2

int main(int argc, char* argv[])
{
 int i;
 for (i=0;i<18353226;i++)
 {
  printf("Hello World!\n");
 }
 return 0;
}
```

当然，这里面省略了很多。我们主要关注的是 cpp 将我们定义的 TOTAL_NUMBER 这个宏，替换成了数字 18353226 了。

这样经过了 cpp 预处理之后的文件，就该交给 gcc 去编译了。

【施工第 2 步——编译】

编译又是什么意思呢？

最初的图纸，也就是没有经过预处理的源代码，那是人写的。一般懂相关语言（比如 C 语言）的人都能看懂。预处理之后的文件，虽然不那么直观了（TOTAL_NUMBER 看着是不是比 18353226 直观？光写个 18353226 还以为是谁的 QQ 号呢），但终究只是做了下替换，还是人类可以看懂的。而经过编译之后的代码是什么样子呢？还以刚才那个 hello.c 做例子，运行：

```
$gcc -S ./hello.c
```

“-S”参数就是告诉 gcc，只对文件进行预处理和编译的步骤。这样最终会得到一个 hello.s 文件，这个文件里就是经过了 cpp 的预处理，并由 gcc 进行编译之后的代码了，大约是下面的样子：

```
    .file   "hello.c"
    .section    .rodata
```

```
.LC0:
	.string "Hello World!"
	.text
.globl main
	.type	main, @function
main:
.LFB0:
	.cfi startproc
	pushq	%rbp
	.cfi def cfa offset 16
	movq	%rsp, %rbp
	.cfi_offset 6, -16
	.cfi_def_cfa_register 6
	subq	$32, %rsp
	movl	%edi, -20(%rbp)
	movq	%rsi, -32(%rbp)
	movl	$0, -4(%rbp)
	jmp .L2
.L3:
	movl	$.LC0, %edi
	call	puts
	addl	$1, -4(%rbp)
.L2:
	cmpl	$2, -4(%rbp)
	jle .L3
	movl	$0, %eax
	leave
	ret
	.cfi endproc
.LFE0:
	.size	main, .-main
	.ident	"GCC: (Ubuntu 4.4.3-4ubuntu5) 4.4.3"
	.section	.note.GNU-stack,"",@progbits
```

这样的代码，就不是普通人类可以看懂的源代码，而是只有终极牛人才能读懂的汇编代码了。汇编代码比较贴近底层的机器码，里面描述的都是一些基本的操作，以机器的思维来描述整个程序。

打个比方。就比如描述切黄瓜的过程，用 C 语言描述出来就像是“将黄瓜切片”，这么一句就搞定了。要是用汇编，那就是：“左手扶住黄瓜，右手拿起刀，移动刀到黄瓜顶部，刀落下，刀抬起。刀向黄瓜后部移动 3 毫米，刀落下，刀抬起。刀再向黄瓜后部移动 3 毫米，刀再落下，刀再抬起。放下刀，走出厨房，走进卧室，找到创可贴，贴在左手食指上……”总之，汇编是一种面向机器的，很复杂的程序设计语言。gcc 的任务就是把 C 语言的源代码转换成贴近机器语言的汇编代码，为下一步 as 的工作做好准备。

【施工第 3 步——汇编】

as 拿到汇编代码后，要对这样的代码再进行处理，得到真正的机器码。这个处理的过程，也叫汇编。

如果说进行汇编之前的汇编代码是终极牛人才能看懂的，那么经过 as 汇编之后，得到的机器码压根就不是人能看懂的了。而且从.c 的源码文件一直到汇编之前的.s 文件都是文本格式的，进行汇编之后就成为二进制的 elf 格式了。这就不是普通的文本编辑器可以打开的了，需要用专门的软件将其转换为 16 进制数据查看。转换过来之后大约就是如图 8.1 所示的样子。

```
0000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
0000010: 0100 3e00 0100 0000 0000 0000 0000 0000  ..>.............
0000020: 0000 0000 0000 0000 4801 0000 0000 0000  ........H.......
0000030: 0000 0000 4000 0000 0000 4000 0d00 0a00  ....@.....@.....
0000040: 5548 89e5 4883 ec20 897d ec48 8975 e0c7  UH..H.. .}.H.u..
0000050: 45fc 0000 0000 eb0e bf00 0000 00e8 0000  E...............
0000060: 0000 8345 fc01 837d fc02 7eec b800 0000  ...E...}..~.....
0000070: 00c9 c300 4865 6c6c 6f20 576f 726c 6421  ....Hello World!
0000080: 0000 4743 433a 2028 5562 756e 7475 2034  ..GCC: (Ubuntu 4
0000090: 2e34 2e33 2d34 7562 756e 7475 3529 2034  .4.3-4ubuntu5) 4
00000a0: 2e34 2e33 0000 0000 1400 0000 0000 0000  .4.3............
00000b0: 017a 5200 0178 1001 1b0c 0708 9001 0000  .zR..x..........
00000c0: 1c00 0000 1c00 0000 0000 0000 3300 0000  ............3...
00000d0: 0041 0e10 4386 020d 0600 0000 0000 0000  .A..C...........
00000e0: 002e 7379 6d74 6162 002e 7374 7274 6162  ..symtab..strtab
00000f0: 002e 7368 7374 7274 6162 002e 7265 6c61  ..shstrtab..rela
0000100: 2e74 6578 7400 2e64 6174 6100 2e62 7373  .text..data..bss
0000110: 002e 726f 6461 7461 002e 636f 6d6d 656e  ..rodata..commen
0000120: 7400 2e6e 6f74 652e 474e 552d 7374 6163  t..note.GNU-stac
0000130: 6b00 2e72 656c 612e 6568 5f66 7261 6d65  k..rela.eh_frame
0000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000150: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

图 8.1　汇编后的文件

如果你想看看汇编后的机器码，那么可以运行命令：

```
$gcc -c hello.c
```

“-c”参数的意思大概您也猜到了，就是告诉 gcc，只进行预处理、编译、汇编这 3 个步骤。这样运行之后会输出一个 hello.o 文件。如果想查看这个文件，运行：

```
$hexdump -C ./hello.o
```

就可以看到一堆乱七八糟数字的机器码了。汇编程序中至少还有些操作的助记符，比如什么 add，mov 之类的。寄存器也是有名字的，比如叫 eax，叫 rbp 等。但是到了机器码，这些都没有了，都换成了各种各样的数字，半句人话都没了。还以切黄瓜为例，要是用机器码来描述，就相当于说：“用 32 号设备扶住 87 号物体，24 号设备拿起 126 号物体，移动 126 号物体到 87 号物体顶部，做 2635 号动作，再做 2636 号动作……”

【施工第 4 步——链接】

好了，现在终于得到机器码了，机器码按说就是可以执行的代码了。但是，这时候的程序还是不能直接执行的，为什么？因为还有 ld 没有出场呢，他的工作叫：链接。

经过预处理、编译、汇编之后的二进制代码，按说已经是机器码了，可以直接运行。但是这里得到的机器码并不完整。就比如刚才说的这个 hello.c 文件，得到的机器码只是针对这个 hello.c 文件里面所写的这么点内容的。而这个 hello.c 里面还调用了 printf()函数，这个函数是在系统的标准输入/输出库里面实现的。这部分负责真正向屏幕上打印字符的机器码如果不包含进来，这个程序怎么可能正常地实现功能呢？所以，就需要把这段标准库中的机器码和我们编译出来的机器码“链接”起来。

而且很多时候，一个程序不是一段机器码，而是由很多段机器码组成的。这些机器码分别保存成很多的.o 文件，最终也需要把它们都“链接”起来才可以运行。

这时候就需要 ld 出场了。

ld 负责把这些机器码组装起来，并且写明了各段代码的地址、从哪里开始执行之类的标记。就像我们造个机器人，脑袋、胳膊、大腿之类的都做好了，ld 就是负责组装的。

经过 ld 组装的程序，就可以运行了。整个编译过程是不是挺复杂的？不过所幸我们的用户并不需要一步一步地手动做这些步骤，只运行 gcc 命令就全搞定了。

8.2　修　理　工

程序编译出来只是第一步。编译好的程序不一定靠谱，可能会有各种各样的错误，这就需要进行调试。调试有很多方法，但肯定离不开调试工具。这回，懒蜗牛同学就要学习调试工具的使用了。

8.2.1　懒蜗牛的日记 B

“2010 年 9 月 18 日　变天

终于又到周末了。今天在自己的机器上学习编程，逐渐开始入门了。这一阵子最大的收获是学会了自学。通过网络、论坛搜索，也能学到不少有用的东西。像学编程吧，开发环境的搭建、循环结构、文字输入甚至创建进程，都尝试过了。看来编程也不是那么的困难嘛。不过我也知道这只是刚刚入门而已，真的写出个能用的程序和随便写个小程序玩玩还是有很大区别的。

今天写了个程序，不知道为什么，总是运行不起来。以前能够运行的程序可以用 printf() 函数打印出变量来看看，这个运行不起来的程序怎么调试呢？”

8.2.2　邪恶的程序

今天起床的时间似乎比平时晚了点，全体起床之后，懒蜗牛同学按部就班地叫来 OO 老先生记录下一些文字，之后又继续去创造他的 rubbish 系列程序去了。

【19 号扰乱秩序】

这一阵子，我们的懒蜗牛真是笔耕不辍——哦，不对，应该是键盘不辍才对——先后制造了 18 个 rubbish 程序。不过很多都被蜗牛删除了，只留有几个：一个是 8 号。因为 8 号比较憨厚，性格温顺，不爱打架。还有一个 16 号。16 号很安静，不爱多说话，有点冷漠，但是做起事情来一丝不苟，严格地执行命令。再就是 17 号很厉害，但是很自大，高傲，总跟别人发生矛盾。他好像和 18 号还有点什么关系，具体的我就不知道了。最后的 18 号是个美眉，长得很可爱，一头金发，本事也不错，我们大家看她都很顺眼。而现在懒蜗牛正在制造 19 号。不知道为什么，我总觉得即将到来的 19 号将是一个邪恶的坏家伙……

没过多长时间，19 号出炉了。只见他起床之后，跑进内存，刚说了几句我是 rubbish 19 号之类的话之后，就开始乱动别人的东西。一会儿要去狐狸的内存空间里拿数据，一会又要往心有灵犀的地盘里存东西。当然，他的这些企图都没有得逞，要是连这样的小流氓都管不了，我还叫内核么。

【内核严明法纪】

我们这个工作间里面的空间管理是很严格的，谁的空间谁用，别人不能乱动。像 19 号这么目无法纪，影响他人工作的软件是不能容忍的。眼看着这邪恶的 19 号，和满工作间无辜受害的软件们，我终于忍无可忍！为了工作间的安宁，为了我稳定内核的荣誉，为了爱与正义，为了世界和平，为了部落，我代表月亮，我，我消灭了你！转眼间，手起刀落，

只听咔嚓一声——整个世界安静了。19 号被我斩为两段，然后我向懒蜗牛汇报：很遗憾，您的程序出现了段错误，就像图 8.2 显示的这样（因为他被砍成两段了，所以错误了）。

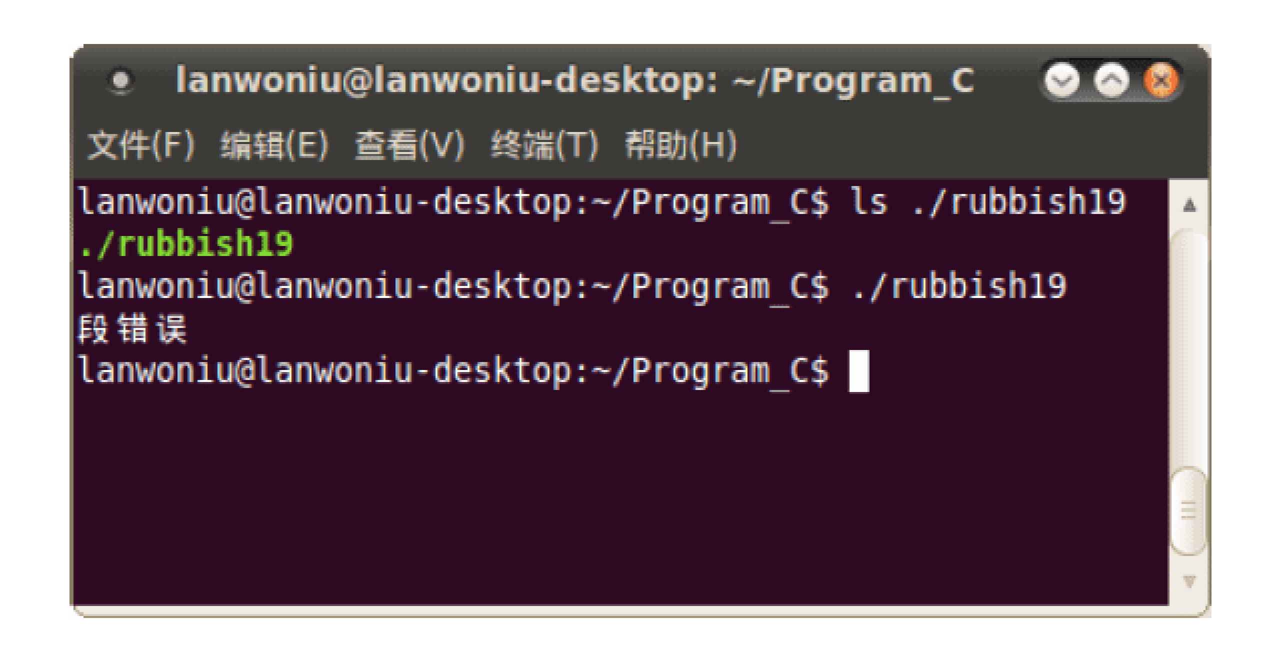


图 8.2　程序出现段错误

懒蜗牛似乎有些不明所以，不知道这个段错误是怎么回事（因为太血腥了，所以我没直说是因为被我剁成两段）。于是就赶紧叫来狐狸上网上查去。通过搜索知道了，段错误的情况有很多（很多种不老实的程序都会被我砍成两段），但大致上都是由于内存指针使用不当引起的。比如没有给指针赋值就去使用，或者虽然赋值但是访问越界等。总之就是动了你不该动的内存就会段错误。

可是到底这个 19 号是如何动了别人的空间的呢？他到底为什么要去访问非法的地址呢？这些情况虽然我们内存里的软件们看得一清二楚，铁证如山，但是懒蜗牛他不知道啊。他没法钻进内存里来看程序是怎么运行的。那么懒蜗牛能有什么办法看清楚 19 号的一举一动呢？这时候就需要我们的软件修理工 GDB 闪亮登场了。

8.2.3　GDB 的简单使用

GDB 是 GNU Debugger 的缩写，也就是 GNU 调试器的意思。它和 GCC 一样，最初也是由 Richard Stallman 设计并实现的。图 8.3 是 GDB 的吉祥物——一条鱼（不要问我为什么，问 Stallman 去）。GDB 是一个字符界面的调试工具。用过 VC 的同学应该知道，在那里面调程序的时候可以进入 debug 模式，能够查看内存、单步执行之类的。我们 Linux 中，每个软件秉承着“只做一件事情，但做到最好”的原则，将调试这件事情交给了 GDB 来完成。

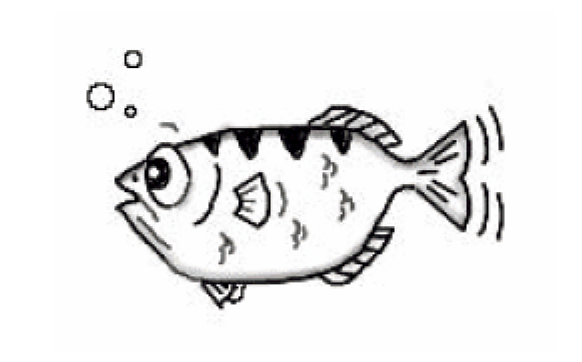


图 8.3　GDB 的吉祥物

【编译出可被调试的程序】

GCC 编译出来的程序可以通过 GDB 来运行，运行的时候，就可以执行设置断点、单步运行、查看变量、查看堆栈等操作。有了 GDB，懒蜗牛同学就可以监视程序在内存里面的一举一动了。

不过 GDB 并不是像狗仔队那样想监视谁就监视谁。像狐狸啦，gedit 这样的成品程序是不能被监视的。要想让某个程序被 GDB 监视，必须在制造他的时候——也就是编译的时候，留出给 GDB 控制的接口来，GDB 才能监视那个程序的一举一动。您看过黑客帝国么？我们机器里的普通程序，就像是里面正常的自然人。而可以被 GDB 调试的程序就像 Matrix 世界中的人一样，脑袋后面有个接口，可以接进去控制。那么怎么给程序装这么个接口呢？很简单，就是在编译的时候加上参数“-g”，类似这样编译：

```
$gcc -g ./rubbish.c input.c readfile.c -o rubbish19_debug
```

懒蜗牛运行了这么一句，就创造出了脑袋后面有接口的 rubbish 19 号。之后就可以叫来 GDB 去调试他了。

提示：没有加“-g”参数编译出来的二进制文件也可以被 GDB 调用并运行，但由于该文件中没有记录机器码与 C 语言源码的对应关系，因此无法进行设置断点、查看变量、查看源码等操作。

【用 GDB 调试程序】

编译出了可调试的程序后，就要叫来 GDB 来运行它，像下面这样：

```
$gdb ./rubbish19_debug
```

于是，GDB 就会接到命令，赶快掏出各种仪器和工具，并把 19 号拖进内存里（这时候 19 号可还没醒哦）。然后从一个大机器上抽出一个长长的电缆，插进 19 号脑袋后面的接口里，一切准备好之后，向懒蜗牛报告：“一切准备就绪，可以开始了。”也就是打印出下面这样的信息：

```
GNU gdb (GDB) 7.1-ubuntu
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/
gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86 64-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/GDB/bugs/>...
Reading symbols from /home/lanwoniu/Program C/rubbish19 debug...done.
(gdb)
```

进入这个界面，就可以进行调试了。当然，要想调试，需要了解一下 GDB 的一些基本的指令。我们简单介绍几个常用的。

- run 命令（或者简写为 r）——这个命令很好理解，就是从头开始运行程序。像刚才懒蜗牛运行“gdb ./rubbish19_debug”后，进入了 GDB 调试环境，但是并没有自动运行起 19 号这个程序，需要 run 一下才行。
- break 命令（或者简写为 b）——这个命令用来设置断点。例如“break 12”的意思就是在程序源码的 12 行设置断点。这样程序运行到这一行就会停下来。
- list 命令（或者简写为 l）——列出当前程序源代码。遇到问题了总得看看源码吧，或者设置个断点什么的也得看着源码才知道应该断在哪行。那就用这个命令查看源码就可以了。
- continue 命令（或者简写为 c）——这是继续执行的意思。程序遇到断点停下来以后，可以用这个命令继续执行下去，直到碰到下一个断点，或者结束。
- print 命令（或者简写为 p）——这个命令用来打印变量的值。比如 print i，意思就是打印出变量 i 当前的值。
- examine 命令（或者简写为 x）——这个命令用于查看指定内存地址中的数据。例如 examine 0x12345678 的意思是查看内存中，地址为 0x12345678 位置所存放的

数据。

🔔**提示**：examine 命令只能查看当前被调试程序能够合法访问的地址。

- next 命令（或者简写为 n）——这是单步执行的命令。程序遇到断点停下来之后，执行这个命令可向下执行一句代码。

懒蜗牛同学好像是之前看书学习过了，虽然据我所知这是他第一次使用 GDB，但是似乎还挺熟练。

进入 GDB 调试环境后，懒蜗牛输入了 r 并且回车。于是，GDB 按下一个电钮，19 号的身体跟着腾地一下站了起来，启动了。插着电缆的 19 号像他上次启动后一样，还是要去骚扰狐狸妹妹，访问人家的内存空间，于是我也只好再次举起屠龙刀，再次将其一刀两断。

GDB 赶快向懒蜗牛报告：19 号同学在按照您的图纸进行某某动作的时候，由于侵占他人内存空间，触犯了内存管理条例第 287 条，因此被处以“断刑”。然后还指出了 19 号的这种行为在懒蜗牛的图纸中所在的位置，也就是代码行数，类似下面这样：

```
Program received signal SIGSEGV, Segmentation fault.
0x000000000040053f in main (argc=1, argv=0x7fffffffe2f8) at ./rubbish.c:9
9         *buffer = 'A';
```

懒蜗牛一看，rubbish.c 文件的第 9 行就出问题了，太伤自尊了，可是这第 9 行也看不出什么不对的来。应该是这个指针有什么问题，还是从头看看程序吧。

于是懒蜗牛输入了 list 命令，GDB 赶紧把 19 号的整个图纸——也就是全部程序的源代码，打印了出来。程序一打印出来，懒蜗牛才看明白：这个 buffer 指针压根就没有初始化嘛，只是做了声明而已，也没给申请内存空间，就这么用，那不出问题才怪。

8.2.4 扩展阅读：内存管理机制

前面向您介绍了懒蜗牛创造的 rubbish 19 号程序，这家伙由于不遵守《Linux 系统内存管理条例》被我干掉了。那么我们 Linux 的内存管理原则是什么呢？都有什么规矩呢？下面就来说说。

咱们说过，一个程序工作的时候需要用到的内存分为几个部分：代码段、数据段、BSS 段和堆栈段。其他的段，基本上程序一启动就确定好了，没什么可说的，也没太多要管理的。唯独这个堆栈里面的堆空间，是程序运行起来之后动态申请的，需要我来管理。

【空间的申请和释放】

堆空间是指程序在起床运行后向我申请来的空间，也是一个程序占用得最多的空间。一个程序如果要想使用工作间里的空间，要向我提出要求，说需要多大多大的一块内存空间——这个过程叫做申请。然后我根据工作间里的情况来分配，告诉他，哪块哪块归你，然后这个程序就去用了。

这时候那块地方就单独给这个程序使用，不许别的程序访问了。如果别的程序胆敢来访问这块空间，就像你去人家偷东西一样，必须依法剁成两段（偷东西没这么大罪过吧……）。

那么这块内存空间分给这个程序使用之后就永远给他了么？想得美！你买房还只有 70 年产权呢，这么珍贵的内存空间怎么可能永久分给一个程序。申请到内存空间的这个程序

在做完了相关的事情，不再需要这块空间的时候，他应该跟我报道，说空间我用完了，这块地方可以再给别的程序用了——这个过程叫做释放。

【内存泄漏】

一个有知识有道德有理想的程序，在他回硬盘睡觉以前应该释放掉所有他申请过的空间。如果遇到哪个不良程序，无德青年申请了很多空间不释放，我也没有办法。因为这不像越界访问那样，访问了别人的地址就是访问了，人赃俱获，无法抵赖。而人家不释放空间，也许是因为他一会儿还需要用这块空间呢，我没法证明他申请的空间用不着了。我唯一可以做的就是在他回硬盘睡觉的时候，检查所有他申请过的空间，如果有没释放的，就强制释放掉——你都睡觉去了，你申请的空间肯定用不着了嘛。

提示： Linux 在程序退出之后将回收程序申请的所有资源，包括内存空间、Socket 连接、设备访问等。

但是如果这个程序是个长时间运行的后台服务程序，并且还不断地申请新的空间而不释放，那就麻烦了。内存空间会被他一点一点地消耗光，这就是最烦人的内存泄漏。在我们软件界，内存泄漏是和瓦斯泄漏同样严重的事故。《Linux 系统内存管理条例》第 3 条明确写着——禁止申请不释放！就在第 4 条禁止抽烟的上面（当然不能抽烟，内存都冒烟了机器还能用么）。

【Windows 与 Linux 对内存使用的不同理念】

其实不光是 Linux，Windows 里的程序一样需要遵守类似的原则，估计他们那里大概也有个什么《Windows 系统内存管理办法》之类的文件吧，反正大家的原理是一样的。不过对于工作间的使用，Windows 和我还是有点不同的。

Windows 总是喜欢尽量留出空间来，好给新起床的程序用。可是我总觉得，作为一个系统，我怎么能知道用户还有什么程序要运行呢？要是没有程序要来了，工作间里还空那么大地方，不让正在工作的程序用，那不是浪费么？所以我还是习惯尽可能地把东西都搬进工作间里。除了程序们申请多少内存就尽可能给多少之外，剩下的部分，我就把一些可能会用到的库、命令等统统都搬进来，能占多少占多少。

有人问：要是你把这里边都占满了，待会儿有程序要进来咋办？很简单啊，我再搬出去呗！

程序要用空间，也不是一下子都占满吧，他也得把他的东西一点点搬进来。他往内存里搬的时候，我就往外搬，不耽误。所以，当某个程序启动，跟我说：“我要 10 平米的地方放东西。”的时候，我就先答应他说：“好，你就往那边那 10 平米放吧。”然后在他往内存里搬数据的时候，我再去给他清理那 10 平米的地方。也可能他要 10 平米，但是暂时只用了 2 平米，那我就先腾出 2 平米来，等他再要我再腾。他们管我这个方法叫 Copy-on-write。

但是 Windows 就不同了。可能是因为他们普遍比较胖的缘故吧，都比较懒，不愿意搬来搬去这么折腾。基本上 Windows 只是在必须用啥东西的时候才把那东西搬到内存里，让内存留出尽可能多的空闲空间。这样，当有程序向他申请内存的时候，就可以用手一指：那块地，归你。然后就不用管了。内存实在不够用的时候就找个比较闲的程序，命令他：你，去硬盘里先忍会儿。

🔔**提示：**图形界面中，可以通过“系统”|“系统管理”|“系统监视器”来查看内存使用情况，如图 8.4 所示。

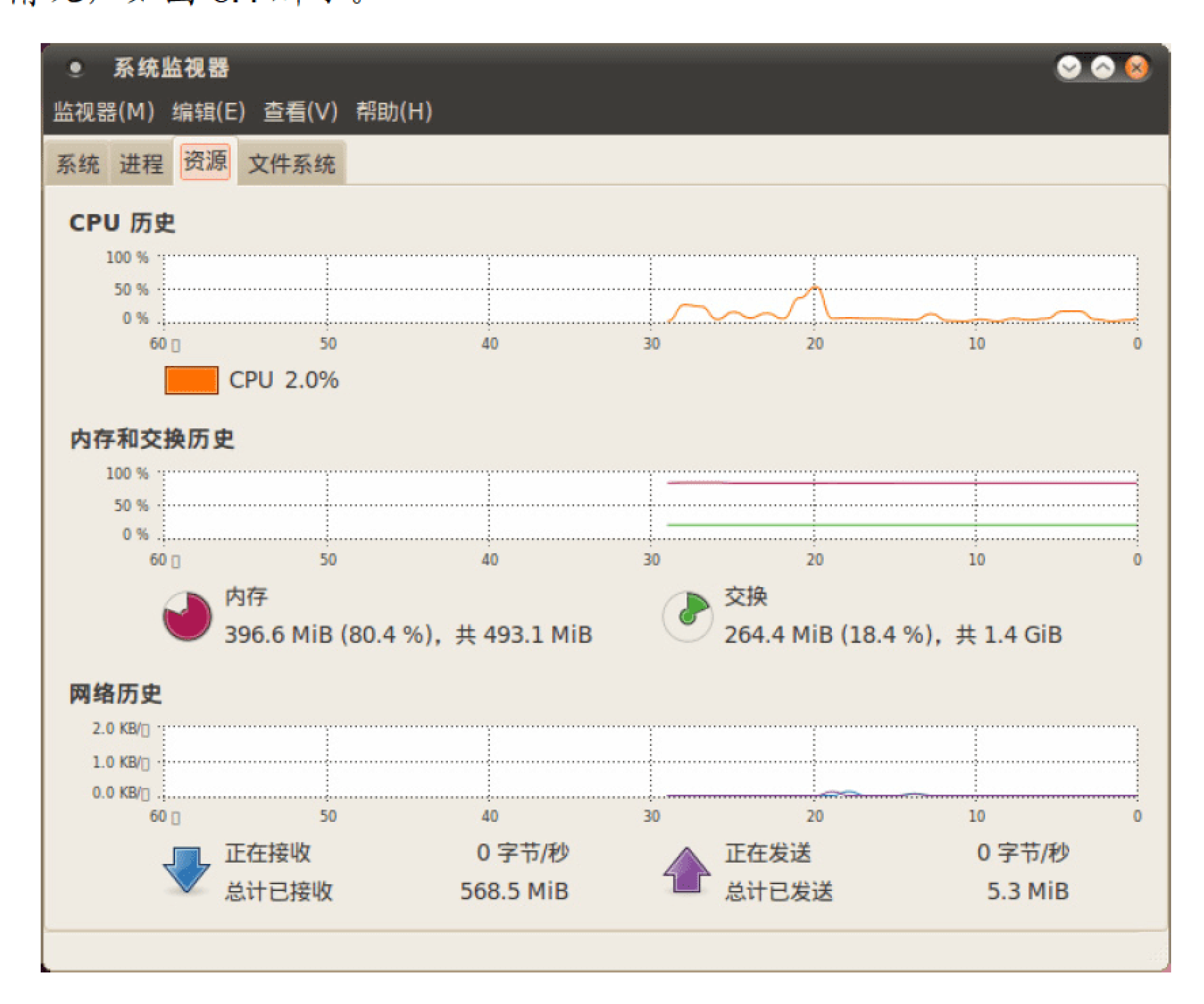
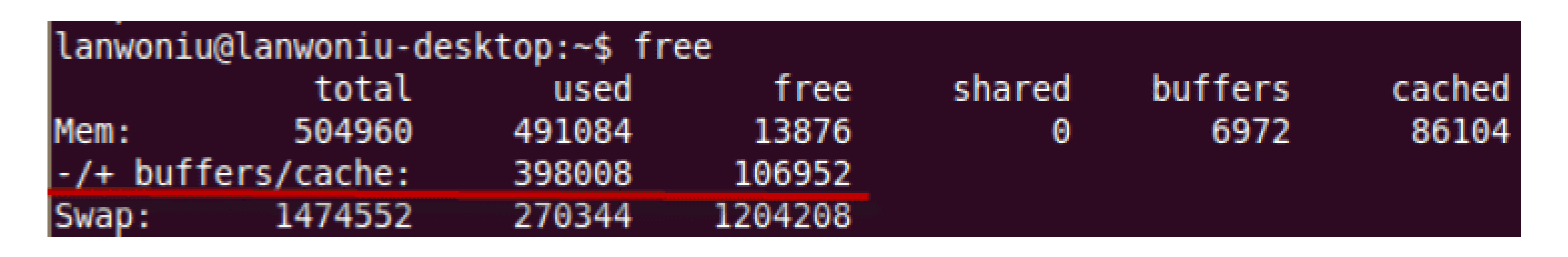


图 8.4　系统监视器

在字符界面中可以通过“free”命令查看内存使用情况。但需注意“free”命令显示的结果中，第 1 行为计算了内核缓冲后的使用率。这个使用率一般很高，多数空闲内存都被用于缓冲。“buffer/cache”一行显示的，才是真正的应用软件所占用的内存，如图 8.5 红线处所示（free 命令显示结果默认单位为 KB）。



```
lanwoniu@lanwoniu-desktop:~$ free
             total       used       free     shared    buffers     cached
Mem:        504960     491084      13876          0       6972      86104
-/+ buffers/cache:     398008     106952
Swap:      1474552     270344    1204208
```

图 8.5　free 命令

8.3　包　工　头

简单的程序直接用 GCC 编译就可以了。但是如果程序越来越大，源码文件越来越多，再手动调用 GCC 来编译就显得费事了。有什么好办法呢？

8.3.1　懒蜗牛的日记 C

“2010 年 10 月 10 日　起风

今天好孤独，节后的只有一天的周末还要独自来加班。领导总是不顾员工的死活，就

像剥削劳动人民的包工头一样。

不过今天加班的时候倒是抽空完善了我的 rubbish 程序。现在这个程序已经有 16 个.c 文件了。每次编译的时候敲命令还真麻烦。还好 Linux 可以记住最近运行的几条命令，不用每次都输入。不过就算这样也挺累的，有没有更好的办法呢？”

8.3.2　越来越多的源码文件

最近硬盘里的 rubbish 越来越多，已经排到 31 号了。虽然这些程序都不大，31 个加在一起也就几兆的大小，可关键是这帮程序大都不着调，不是段错误就是内存泄漏，要不就是乱访问设备。这么多不着调的程序在内存里折腾，指不定什么时候就出事了。

【文件太多，一起编译费时费力】

这些 rubbish 程序，除了不着调以外，个头越来越大，源码也越来越复杂。从 rubbish 1 号只有一个.c 文件，发展到现在，rubbish 31 号一共有 10 多个源码文件。每次懒蜗牛同学编译 rubbish 31 号的时候，都要这样：

```
$gcc rubbish.c input.c display.c think.c ai.c mem.c disk.c file.c speak.c
banana.c apple.c i.c love.c you.c pentax.c km.c -o rubbish31
```

每次看着他输入这么长一串的命令，我真替他累得慌。并且每次懒蜗牛这么一编译，施工队那哥儿几个就都得跑过来，把这一大堆.c 挨个打开，开始从头施工，一个一个编译。有时候懒蜗牛只是在 ai.c 里面修改了很少的一点，编译的时候施工队的同志们也要从头到尾地重新编译每一个文件。

这就好像盖个楼，完工之后开发商说：一楼这个大门的门把手图纸上画错了，应该用圆的，怎么画成方的了？改了吧。施工方一听，赶紧下令：“图纸画错啦！把楼炸了重新盖！”虽然这样对于拉动 GDP 发展有很好的作用，但毕竟属于精神不正常的范畴。

【分别编译，一起链接】

那么我们的 GCC 施工队为什么做这种很抽风的事呢？这不怪施工队，这是因为懒蜗牛输入的命令就是让他们拆了重盖的意思（也就是让他们从头开始编译），他们只是严格执行命令而已。实际上完全不必这样，那些.c 文件中，改了哪个只编译哪个就可以。那应该怎么操作呢？

（1）首先，源码写完了之后先各自编译成模块，运行：

```
$gcc -c rubbish.c input.c display.c think.c ai.c mem.c disk.c file.c speak.c
banana.c apple.c i.c love.c you.c pentax.c km.c
```

这个命令的意思就是把这些.c 文件各自编译成.o 文件，如 rubbish.o、input.o 等。

（2）要获得最终的二进制文件，只要再运行：

```
$gcc rubbish.o input.o display.o think.o ai.o mem.o disk.o file.o speak.o
banana.o apple.o i.o love.o you.o pentax.o km.o -o rubbish31
```

就生成了最终的 rubbish 31 号。

（3）那么之后如果修改了某个源文件——比如修改了 ai.c 文件，只需要重新编译这一个文件就行了，就运行：

```
$gcc -c ai.c
```

这样就编译出了新的 ai.o。

（4）得到新的 ai.o 之后再运行：

```
$gcc rubbish.o input.o display.o think.o ai.o mem.o disk.o file.o speak.o
banana.o apple.o i.o love.o you.o pentax.o km.o -o rubbish31
```

重新链接一下，新的 rubbish 31 号就完成了。这样就省去了重新编译那些没有改动过的文件的时间。

8.3.3　make 的机制

不过这样编译虽然节省了编译的时间，但是敲起命令来也挺麻烦的。有没有更方便的方法呢？当然有。

【大项目需要规划】

其实用户完全不必每次都敲一大长串的 gcc 命令来编译程序。如果是那样，我们 Linux 内核有上千个文件，要是编译一次，光敲命令就得敲一上午。

那个 GCC 施工队毕竟只是个施工队，你要是盖个小厨房，垒个猪圈，这样的小东西直接找他们没问题。直接一编译：“gcc 砖头 -o 猪圈”就出来了。可如果要盖个 CBD 商圈，里边什么银行、商场、写字楼、炸油条的、卖臭豆腐的、修理自行车的等，一应俱全，这么大的一个工程，你光叫个施工队来肯定搞不定。这得有人进行合理的统筹规划，设计施工方案，然后再让施工队去具体施工。这个规划的人谁呢？按照懒蜗牛现在的做法，这个规划人就是懒蜗牛自己，但他自己又没这本事，怎么办呢？这时候他就需要专业的规划人，能够指挥施工队的包工头——make。

【make 的重要作用】

make 也是一个程序，像上面说的一样，他就是负责控制整个施工过程的（也就是编译过程）。对于比较小的程序，就一两个.c 文件，根本用不着 make 出马，GCC 施工队去编译就行了，因为源文件的结构关系不是很复杂。可是对于稍大一点的程序，像狐狸妹妹、心有灵犀、星爷啊，基本上所有常用的软件，都足够复杂到需要 make 来对编译过程进行管理。

如果软件大了，编译的时候就不能简单地把一大堆.c 的源文件统统一次性编译成一个二进制文件，这种方法太粗鲁了。应该像上面介绍的那样，把一堆.c 文件编译成一堆.o 文件，然后再把.o 文件链接成一个成品的二进制文件。有改动的时候只更新单个的.o 文件就可以。

提示： 不一定非要把每一个.c 文件编译出对应的.o 文件，可以几个.c 文件生成一个.o，一切根据具体代码的需求来设计。

但是这个过程如果由人类来负责，就不那么靠谱了。他们的大脑不可靠，不一定能记清楚这次改了哪些文件，应该更新哪个.o 文件。于是，make 就义无反顾地挑起了这个重要的担子。当然 make 也不能靠凭空的想象就来指导包工队干活，什么事情总得有个规划，make 也需要一份施工的规划书，这份规划书就是 Makefile。

8.3.4　Makefile 的基本格式

Makefile，顾名思义，就是 make 用的 file。这就相当于一份施工的规划，上面写着整个工程分为几个模块，先用哪几个文件编译成一个什么什么.o，再用哪几个文件编译出一个.o，再怎么怎么一链接，最后得到编译好的二进制程序。

make 就根据这份文件来指导 GCC 他们进行施工。当有某个.c 文件被修改之后，make 能够根据文件的修改时间智能地判断出哪些模块需要重新编译，重新链接，然后就去让 GCC 重新编译那些改过的文件，最终生成新的二进制程序。

有了 make，写好了 Makefile 文件，就省去了用户敲一大堆编译命令的烦恼。只要敲一个 make 命令，其他的，就交给 make 去做吧。他办事，你放心。

【简单的 Makefile 示例】

比如说，有这么一个工程，包含了 3 个文件（咱就不拿懒蜗牛同学的 rubbish 系列打比方了，源文件太多，还乱）。分别是：main.c、part1.c、part2.c。那么我们就可以试着写出一个用于编译这个工程的 Makefile 如下：

```
all:main.o part1.o part2.o
    gcc main.o part1.o part2.o -o mybin

main.o: main.c
    gcc -c main.c

part1.o: part1.c
    gcc -c part1.c

part2.o: part2.c
    gcc -c part2.c
```

乍一看可能您不太明白，没关系，咱们慢慢说。当用户运行 make 命令的时候，make 就会来到当前目录下，首先在这个目录里查找 Makefile 文件，如果没有，就找 makefile 文件（Linux 区分大小写嘛）。如果都没找到，就报错。如果找到了，那就打开 Makefile 看看该干些什么。

【Makefile 中的基本书写格式】

首先我们看到，这个 Makefile 大致分成了 4 段，每段的格式都差不多。我们把它总结为这样的格式：

```
目标：原料
<Tab>加工方法
```

对照着来看，先看第 1 段：

```
all:main.o part1.o part2.o
    gcc main.o part1.o part2.o -o mybin
```

这段的“目标”是 all，all 是一个 make 的关键字，当用户运行 make 并不加任何参数的时候，make 就会来 Makefile 里找到目标为 all 的这一段，并且从这里开始干活。

然后，这段的“原料”是 main.o、part1.o、part2.o 这 3 个文件。也就是说，要想达到 all 目标，需要先有这 3 个文件。于是 make 就会查找现在是否有这 3 个文件，一看——

没有！

没有没关系，make 会继续往下找，这回的目标，就是查找怎么才能搞到 main.o 文件，结果一找，还真有，第 2 段就是：

```
main.o: main.c
    gcc -c main.c
```

这段的“目标”就是 make 正要找的 main.o，于是赶紧看看原料，是 main.c 文件。这个文件已经有了。那么怎么用这个原料加工成 main.o？看方法：gcc –c main.c。哦，原来运行这条命令就行了啊，于是 make 就会去调用 gcc，来编译出 main.o。

提示：“加工方法”一行的前面有且必须有一个 Tab 制表符，不能顶格写，也不能用空格代替 Tab。

有了 main.o，make 再回去看第 1 段，发现还需要 part1.o，part2.o，跟 main.o 的处理方法一样，根据 3、4 两段就可以编译出来了。原料都齐全了之后，make 就再根据第 1 段的“加工方法”运行“gcc main.o part1.o part2.o -o mybin”，生成了最终的目标。

另外，刚才说了，如果不加参数，make 就去找“目标”是 all 的段落。其实用户也可以通过参数指定 make 的目标，比如用户运行：

```
$make main.o
```

意思就是去完成 Makefile 里面，“目标”是 main.o 的那段任务。于是 make 就根据 Makefile 里的记录，只编译出一个 main.o 来。

提示：如果 Makefile 中没有“目标”为 all 的段落，并且运行 make 没有指定参数，则 make 会执行 Makefile 中的第 1 个段落，无论目标是什么。

【根据时间决定动作】

当某一个.c 被修改了之后，用户应该如何编译呢？简单，还是只执行 make 就可以了。剩下的事情，就交给 make 去做吧。

make 会检查每一个“目标”和“原料”的最后修改时间。比如 part1.c 文件被修改了，那么 make 就会发现，part1.o 的创建时间要早于 part1.c 的最后修改时间，这说明 part1.o 需要被重新编译。于是他就会按照这一段的“加工方法”，再次运行 gcc -c part1.c，来编译出新的 part1.o。

那么现在 part1.o 的创建时间又比最终的 mybin 新了，于是 make 又根据第 1 段的加工方法执行了 gcc main.o part1.o part2.o -o mybin，把新的 part1.o 和没有变动的 main.o、part2.o 链接成了新的 mybin 文件。

【多种多样的“加工方法”】

上面的例子里，加工方法一行基本都是编译或链接命令。其实，宪法里并没有规定“加工方法”必须是跟 gcc 有关的。其实加工方法这一行写任何命令都可以，并且不一定只写一行，写几行都可以，只要是挨着就行。比如一般的 Makefile 里都会有类似下面这样一段，用于清理编译结果：

```
clean:
    rm ./*.o
```

```
    rm mybin
```

这一段的目标是 clean，没有原料。用户执行：make clean 的时候，make 就会找到这一段，并且发现不需要原料，于是直接执行“加工方法”，于是就删除了所有.o 文件和最终的编译目标文件。于是整个源代码的目录里恢复到了编译之前的样子。

与此类似的还有 make install，一般就是下面这个样子：

```
install:mybin
    cp ./mybin /usr/bin/
```

也就是在确认当前目录下有 mybin 这个编译好的文件后，把这个文件复制到系统中的相应目录，就完成了安装。

8.4　分　析　师

一说到 make，很多人都记得编译源码包的时候，经常在进行 make 之前还要运行“./configure”命令，这个命令又是干什么的呢？

8.4.1　懒蜗牛的日记 D

“2010 年 11 月 15 日 晴

这个 make 工具果然方便啊。听说 Windows 下的 VC 其实也使用了类似的东西，只不过把它们都用图形界面封装了起来，所以我看不到了。看来还是在 Linux 下学习编程才能了解到一些本质的东西呀。

我还看到很多以源码包发布的软件，都会有一个 configure 脚本，成功地运行了这个脚本以后才能去运行 make 命令。这个脚本又是干什么的呢？”

8.4.2　源码软件的平台依赖

懒蜗牛同学最近是越来越不着调了，竟然想把他最新写的那个 rubbish 1115 号发布到网上去！祸害我们一个系统还不够，还要残害多少青春懵懂的 Ubuntu 啊。

【rubbish 1115 号——放到哪都是个祸害】

其实那个 rubbish 1115 号也干不了啥正经事，主要就是能陪懒蜗牛同学玩“猜拳”的游戏，所以才这么受宠。说来也怪，我们软件源里面有那么多游戏，懒蜗牛都没兴趣，不知道为什么就对这么个石头、剪子、布的随机函数情有独钟。不过也难说，毕竟是他自己编的嘛，谁的孩子谁不爱呢。可是您自己喜欢那就偷偷摸摸自己玩就行了，干嘛非要发布到网上让他去祸害别的电脑呢？

这家伙经常申请了内存不释放，有时候还假死，如果是不明所以的人用了这个程序，没准还抱怨我们 Linux 系统不稳定呢。当然，发牢骚归发牢骚，懒蜗牛的命令我们还是得执行，Firefox 就正忙着把 rubbish 1115 号传到网上去呢。

过了一会儿，狐狸妹妹忍着笑就过来了：“你知道懒蜗牛怎么发布他那个程序么？懒蜗牛直接把编译出来的二进制文件贴到了论坛里面。哈哈，他以为这样直接就能运行呢。

笑死了。”

【二进制程序——不是放到哪都能运行】

嗯，看来懒蜗牛同学还有很长的路要走啊。这个二进制文件看上去就是单一的文件，但其实他运行起来是需要很多库文件来协助的，不是拿到哪都能运行的。在他们 Windows 界其实也是这样：Windows 98 的程序直接拿到 Windows XP 下不一定能运行；Windows XP 的程序也有很多装不到 Windows 7 上。但是由于他们的版本比较少，而且系统的各种库和接口等都比较统一，所以也有不少的绿色程序直接复制到系统里就能运行。因此很多人觉得程序就是一个 EXE 文件，复制到哪里都可以运行。

懒蜗牛写的这个程序，如果复制到一个和我们相同的系统上，肯定可以运行（也肯定可以造成内存泄漏和假死，哼哼）。可是不一定别人的系统就跟你的系统一模一样啊！尤其我们 Linux，发行版五花八门，就算相同的发行版，版本不一样，也不一定能行。如果系统里没有这个程序所依赖的那些库，这个程序肯定是运行不起来的。要想知道一个程序依赖于哪些库，可以用 ldd 命令来查看。

【查看依赖关系——看他到底在哪能运行】

我趁懒蜗牛不注意的时候叫来 ldd，运行了一下 ldd ./rubbish1115，看看这个软件都依赖什么。ldd 向我汇报如下：

```
$ ldd ./rubbish1115
     linux-vdso.so.1 =>  (0x00007fff31dff000)
     libc.so.6 => /lib/libc.so.6 (0x00007ff61215c000)
     /lib64/ld-linux-x86-64.so.2 (0x00007ff6124cc000)
```

看来依赖的还不是很多，算是最基本的了，可照样不能随便放到别的系统上运行啊。比如这个“ld-linux-x86-64.so.2”，明显只有 64 位系统才可以。“libc.so.6”虽然是个系统就有，但是版本也得合适，不合适也不行。这也是为什么 Linux 上发布的软件好多都是源码包的原因，因为系统环境实在是各式各样，还是把源码放到目标系统上编译一遍更方便。然而我这些话也就跟您说说，懒蜗牛听不见我说话的，所以他还是把那个 rubbish 金刚直接贴到网上了。

8.4.3 一个标准的源码包安装过程

几天之后，懒蜗牛就收到了应有的回应。

“这个程序在我这运行不了啊？”

“楼主我用 SUSE 啊，你编译出个 SUSE 版本的来吧。”

“楼主再好好看看吧，我这里运行不了。”

“我是 32 位机啊，运行不了这 64 位的。楼主提供个 32 位的版本吧。”

……

懒蜗牛同学终于意识到二进制文件不是放到哪都能执行的，可要让懒蜗牛去给每个人编译一个针对他们系统的版本也是不大可能的。好吧，那就发布源码！

可是问题又来了，源码发布的软件包应该是什么样子呢？为了避免再次体现自己的无知，懒蜗牛从网上下载了一个 lynx2.8.7.tar.gz 软件包。这个 Lynx 是一个字符界面的浏览器，懒蜗牛倒不是想用它，主要是这个软件体积不大，依赖也不多，正好拿来体验一下源码包

的安装。

【要装源码包，先打开看看】

像“.tar.gz”这样的源码包是下载软件时最经常遇到的了。另外还有“.tar.bz2”格式，跟“.tar.gz”的没什么区别，只是压缩格式不同而已。就类似于一个是 RAR，一个是 ZIP。那么这样的软件包怎么装呢？当然是先把包解开再看了，得先解开压缩包看看里面是什么内容才能知道怎么装啊，就像我问你 RAR 包怎么装，你能知道么？

“.tar.gz”格式的文件，就像是一个邮局寄来的包裹。你收到一个包裹后怎么办？当然是先打开啦！先找剪子、小刀之类的工具把包裹拆开，然后看看里面有什么东西，根据里面东西的不同来决定怎么处理。里面要是家里寄来的松子核桃之类的特产，就赶快吃了；要是比较难吃的松子核桃什么的，就跟同事分着吃了；要是部手机，就赶快拿出来试试；要是下面还有把手枪，就赶紧拿刚才那手机报警。

【解压 TAR 包的工具】

这些大概不用我说，智力正常的人都应该知道怎么做。其实 TAR 包也是如此。拿到一个 TAR 包之后，先用你的工具把 TAR 包拆开。工具是啥？有道是解铃还须系铃人，TAR 打的包，当然还用 TAR 来解了。一般解压一个 TAR 包的命令是：

```
$tar -xzvf xxxx.tar.gz
```

这里，xzvf 是 tar 命令的参数，我们分别解释一下。

- x——参数 x 意味着要做解包的动作，与之相反的是 c，也就是打包的意思。
- z——意思是这个包是用 gzip 压缩过的，需要先调用 gzip 解压。
- v——显示解压的过程，也就是打印出解压出来的文件。如果没有这个参数，则在解压过程中没有任何输出。
- f——指明要解压的文件。

另外还有一些常用的参数，也顺道介绍一下。

- j——意思是这个包是用 bzip 压缩过的（也就是.tar.bz2 格式），需要先调用 bzip2 解压。
- c——这个参数的意思是要做打包的动作，和 x 参数相反。
- C——注意这个参数是大写的 C，用于将解压缩后的文件存放到指定目录。如果没有这个参数，则默认解压到当前目录。

当然，也可以用那个叫做文档管理器的家伙，他的中文名字叫归档管理器，他的英文名字叫 file-roller。不过其实他只是个负责用图形界面和用户交流的家伙，真正干活的还得是 tar。

提示： tar 命令后面的参数顺序并没有特定要求，但要确保文件名紧跟在 f 参数后。如“tar –vzxf xxx.tar.gz”，“tar –xvzf xxx.tar.gz”都是正确写法。

TAR 包解开后，一般会得到一个目录，里面有很多的文件。然后干什么呢？有的同学记起来了，看看里面的东西啊。

一般包里面应该有个 README 文件。文件里写着这个软件是干什么用的、怎么安装、怎么用、作者是谁、何方人士、爱吃什么、身高多少、腰围裤长等信息。也可能安装的方法写在一个叫做 INSALL 的文件里。总之，应该有相应的文档文件来告诉你这个软件怎么

装。不过也有时候软件的作者不厚道，或者忘性大，没有写 README 或者 INSTALL 文件。或者文件是有，但是没说清楚到底怎么装，那怎么办呢？只好去给作者写个 E-mail 鄙视他了。

8.4.4 configure 的作用

懒蜗牛把下载来的软件包移动到了他的家目录下，然后运行：

```
$tar xzvf ./lynx2.8.7.tar.gz
```

把这个压缩包解压了出来。解压之后是一个目录，叫做 lynx2-8-7。懒蜗牛继续运行：

```
$cd lynx2-8-7
```

这样就进入了刚刚解压出来的目录里面，用 ls 看了一下，发现有很多文件：

```
$ ls
ABOUT-NLS      configure        install-sh        makefile.in   samples
aclocal.m4     configure.in     lib               makefile.msc  scripts
AUTHORS        COPYHEADER       LYHelp.hin        makelynx.bat  src
BUILD          COPYHEADER.asc   LYMessages en.h   make-msc.bat  test
build.com      COPYING          lynx.cfg          makew32.bat   userdefs.h
CHANGES        COPYING.asc      lynx help         mkdirs.sh     VMSPrint.com
clean.com      descrip.mms      lynx.hlp          PACKAGE       WWW
config.guess   docs             lynx.man          po
config.hin     fixed512.com     lynx.rsp          PROBLEMS
config.sub     INSTALLATION     makefile.bcb      README
```

其中有个叫做 INSTALLATION 的文件，里面很显然应该写着安装方法。于是懒蜗牛同学打开这个文件看了看，内容很多。不过关于安装，他看到这么几行：

```
If you are on a UNIX platform, the easiest way to build Lynx is to type:
       ./configure
   and
       make
```

看来是运行这样两个命令就可以了。于是，懒蜗牛同学按照说明，运行了：

```
$./configure
```

这个 configure 是干什么的呢？

【施工之前，先勘察好环境】

我们知道 GCC 施工队听 make 包工头的指挥，make 包工头根据 Makefile 安排工作。这样，如果想把一堆源码编译成二进制的程序，只要执行一下 make。执行之后 make 会在当前目录下寻找 Makefile，然后按照上面写的方案，指挥施工队：在这盖个大裤衩，在那盖个水煮蛋，再在中间垒个鸟窝。然后施工队按照命令一点点施工，直到最终完成任务。

然而事情有时候并不是那么简单。没准包工头 make 下达建设大裤衩命令之后，施工队回来报告：这地方挨着鞭炮厂啊，盖大裤衩还不烧着了？包工头说：那先盖水煮蛋吧。施工队又报告：这地方常年干旱，地下水位也低，这点水连泡面都不够，别说煮蛋了。包工头只好说：那就先盖那鸟窝，总行了吧？施工队再说：鸟窝倒是能盖，就是这地方不通天然气，点不着窝里那火炬啊。

【分析师出马】

遇到这些问题，都是由于开工之前没有对施工的环境、现有的材料进行合理分析导致的。那么我们的这个 configure，就是能够解决这种问题的一名分析师。

configure 跟 make 不一样，他并不是常驻在我这里的软件，而是每个源码发行的软件自带的一个脚本。简单点说，铁打的 make 只有一个，流水的 configure 每个软件一个。

有了 configure 之后，编译软件的步骤就多了一步——./configure。让这个分析师首先开始工作，他会检查当地的情况，有什么材料、什么库、什么编译器之类的，都检查一遍，然后因地制宜地设计一份 Makefile。如果有足够的水，才允许煮蛋；有远离火种的安全空间才能晾裤衩等。如果条件不满足，configure 就会报告错误，告诉用户这里缺少什么，等用户想办法弄齐了再来编译。如果条件满足可以施工，configure 就会出一份 Makefile。注意，一般 configure 调查前，目录下是没有 Makefile 文件的（当然，没有 configure 的情况另说）。

懒蜗牛运行./configure 之后，得到了如下输出结果：

```
$ ./configure
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking target system type... x86_64-unknown-linux-gnu
Configuring for linux-gnu
checking for DESTDIR...
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
……//此处省略若干行
checking for _nc_free_and_exit... yes
checking for _nc_freeall... yes
srcdir is .
updating cache config.cache
configure: creating ./config.status
config.status: creating makefile
config.status: creating WWW/Library/Implementation/makefile
config.status: creating src/makefile
config.status: creating src/chrtrans/makefile
config.status: creating lynx_cfg.h
lanwoniu@lanwoniu-desktop:~/lynx2-8-7$
```

这中间省去了几百行，无非就是“checking for xxxxxx... yes/no”这样的格式。这些输出的意思，就是说 configure 在对我们系统进行检查，报告有什么材料，没有什么材料。如果不是必需的材料，没有也就没有了。如果是必需的东西没有，那么 confiugre 就会报错并停止。

最终，我们这个系统里的东西还比较全，configure 发现可以施工，于是就生成了 Makefile 文件。

提示： Makefile 文件中可以引用另一个 Makefile 文件，因此一个软件工程中，经常可以看到不同源码目录下都有一个 Makefile 文件。

生成了 Makefile 文件，于是懒蜗牛同学就运行 make 命令：

```
$make
```

make 开始工作，指挥着 GCC 施工队进行编译。由于软件很小，马上就编译完了。编译完成之后，就在当前目录下生成了一个叫做 lynx 的二进制文件。不过如果就这么放在这，运行起来很不方便，所以懒蜗牛同学继续运行了：

```
$make install
```

这才把这个软件安装在了我们系统里。

提示：如果要删除源码安装的软件，可以在源码目录下运行“make uninstall”。作为一个标准的 GNU 软件，生成的 makefile 中应该都包含有 uninstall 的定义，但实际有一些不规范的软件，没有提供 uninstall 的方法，就只能手动删除了。

【照猫画虎】

有了这么一番感受之后，懒蜗牛知道了一个源码发布的软件包的大概样子。于是照着人家这个软件包，对比一下自己的这个软件。一看，Makefile 是现成的，只要再增加一个 configure 脚本，检查一下系统中有没有 gcc 之类的编译工具及库文件等就可以了。

要写这么个脚本，需要用到 Shell 脚本编程的知识。懒蜗牛之前虽然学过一点，但那都只是些皮毛。现在要真去写 configure 脚本，还真有点力不从心。怎么办呢？对，照猫画虎！看看那个 Lynx 的 configure 怎么写的，跟着学就好了。懒蜗牛想得挺好，叫来 gedit 打开 configure 一看就傻了——洋洋洒洒连注释带语句一共 36379 行！这得看到哪年啊！

8.4.5 扩展阅读：黄金搭档——tar 和 gzip

前面介绍到 tar 命令，基本上每个 Linux 系统都会带着这个软件，我这里也是。这个软件是干什么的呢？

tar 就是个打包裹的，不过他可不是邮递公司的那种，不会把打好的包扔来扔去。他的能力有点像 Windows 7 那里的 WinZip，他能把很多文件和目录收拾在一起，打成一个包裹，也就是生成一个 tar 包文件。

提示：过去的计算机使用磁带作为长期存储数据的介质（现在依然有使用磁带作为存储介质的场合），tar 命令最初就用于将数据打包存储在磁带上。tar 就是 Tape Archive（磁带归档）的缩写。

可是跟 WinZip 不一样的是，tar 只管打包，不管压缩。原来那些零碎的小文件有多大，打成 tar 包之后还是多大，只是变成一个整个的文件了而已。有人说，那我想压缩怎么办？别急，我这里还有另一个软件，叫 gzip。这个软件就是专门负责压缩和解压缩的，但是他只能压缩单个文件，不能像 WinZip 那样能压缩一个目录里的很多文件。

这样，tar 和 gzip 就成黄金搭档了。要想实现 WinZip 那样的功能，就得 tar 和 gzip 联手协作。比如有个目录叫 aaaa，里面有好几十个文件，总共有 10 MB 大小。想要压成 ZIP 那样的压缩包，那就先让 tar 出手，把 aaaa 目录打成一个包裹文件——因为 gzip 只能压缩单个文件嘛。这样 tar 就把这个目录打成了 aaaa.tar 文件，这个文件还是 10 MB 大。然后由 gzip 出场，把这个文件压缩，压缩完了得标明一下啊，所以就又把文件名改了，叫做 aaaa.tar.gz，表示这个文件经过了 gzip 压缩。这时候这个文件就小了，可能 5 MB，也可能

7 MB。有时候还有叫 xxx.tgz 的包，也是一个意思，只是把.tar.gz 的扩展名合并了而已。

8.5 规　划　局

configure 脚本也好，Makefile 文件也好，其写法都是有一定规律可循的。并且他们的内容都是有一定复杂度的。对于有规律还复杂的东西，就可以想办法让程序自动实现。

8.5.1　懒蜗牛的日记 E

“2010 年 12 月 20 日 回冷

总算是把代码发布到网上了。很多热心的网友提出了不错的建议和意见。才发现我写出来的程序真的很白痴。经过了这么一个过程，确实在编程方面长进了不少，了解了很多以前不了解的事情。

还有件最不了解的事，就是 configure 脚本。好几万行的代码啊！牛人们是怎么写出来这么复杂的脚本的？而且好像很多源码包里的 configure 脚本都差不多，难道都是一个专门给别人写脚本的大牛写的？不解中……”

8.5.2　自动生成的 configure 脚本

懒蜗牛的 rubbish 1115 号放到网上去之后，网络上的众多牛人们，为他修改了很多问题，把他调教得规规矩矩的。现在的 rubbish 1115 号，也不浪费内存了，也不乱改文件了，也不死机了，腰不酸了，背不疼了，现在我们都开始喜欢这个家伙了。他好，我们也好。这大概就是开源的力量吧。我们现在都不好意思叫他 rubbish 了，直接叫 1115 号。

【3 万行的脚本真不是人写的】

1115 号发布前的那段时间，我们几个软件都很纠结，担心懒蜗牛运行他，担心系统被他搞坏。那时候懒蜗牛也很纠结，纠结的是怎么能够把他发布到网上去。

那时候懒蜗牛整天研究 Shell 编程的技术，天天对着那 3 万多行的 configure 代码，出神地看着，嘴里念叨着：“这是哪位神仙大姐写的脚本啊……这么多行得写多长时间啊……”直到有一天，他终于将眼睛聚焦在了 configure 文件前面的那段注释中的一句话：

```
#Generated by Autoconf 2.52.20081225.
```

懒蜗牛顿时如醍醐灌顶一般，看着这一行注释，看着这个“Autoconf”，心里反复地呼喊，声音越来越强烈，直到终于爆发，脱口而出：“原来这脚本是用软件自动生成的啊！”顿悟之后懒蜗牛立刻叫来狐狸，本着“内事不明问老婆，外事不明问 Google”的宗旨，直奔 www.google.com 而去。

【3 万行的脚本到底是谁写的】

一番查找后，懒蜗牛终于大致了解了 Autoconf 这个软件。

咱说 gcc、cpp、as 和 ld 他们 4 个命令就像施工队，make 就是包工头，configure 就是

分析师，那这个 Autoconf 大概就是市政规划局了。有了他，什么 Makefile，configure 脚本，全都不用自己写，都由他一手代办。

规划局的工作，就是根据源代码的结构和组成，来决定如何根据环境，因地制宜、就地取材地施工，最终派出一个专门的分析师——也就是 configure。之后在安装的时候，configure 就可以根据目标系统的环境及既定的几套施工方案，来写出合适的 Makefile，再交给那里的 make 去指导施工。

8.5.3 规划局的成员组成

虽然软件叫做 Autoconf，但其实并不是只有他一个人。既然叫做规划局，那就不可能是一个人，你见过哪个地方的规划局就一个局长了？他们这规划局成员有 4 个：Aclocal、Autoconf、Automake、Autoscan。要想自动创建 Makefile 和 configure 脚本，就得跟这哥儿 4 个说。虽然可能你已经自己写了 Makefile 了，只是缺少 configure。但你写的那个不行，他们向来是买一送一，搭配销售。你写的 Makefile 是没用的，必须得用他们创作的 configure 和 Makefile，具体情况咱们待会儿再说。

这 4 个人各有各的工作，各司其职：Autoscan 负责检查源码目录结构，看看都有哪些需要编译的文件；Aclocal 用于检测一些编译环境相关的内容，例如使用哪个编译器；Autoconf 负责生成 configure 脚本；Automake 负责生成 Makefile 的蓝本——makefile.in。

8.5.4 图纸审查

经过一番查找，懒蜗牛同学已经了解了 Autoconf 这一组软件的使用了，现在他要开始为他的程序加入 configure 脚本和 Makefile 了。

首先，懒蜗牛来到存放 1115 号源码的那个目录，目录里现在有 main.c、board.c、ai.c、board.h 和 ai.h 几个文件（1115 号已经被网络上的热心爱好者们改装得很精简了）。然后懒蜗牛运行了这个命令：

```
$autoscan
```

那么“autoscan”这个命令是干什么呢？

【Autoscan 的职能】

Autoscan 是负责初步审查项目的。你的工程图纸画好了，得先拿给他看。他看了一遍之后，会给你写个报告。怎么还写报告？当然了，规划局嘛，审批个这处理个那的，不都是部门之间报告来、报告去的么。Autoscan 写的这个报告叫做 confiugre.scan。

但是这个 Autoscan 写的 confiugre.scan 报告，基本上是驴唇不对马嘴。Autoscan 这家伙，要论本事，抬举他点说是一般。图纸他都不一定看得懂。所以多数情况下，还得动手改改。改过的报告还得改个名字，叫做 configure.in。

正说着，只见 Autoscan 同学大摇大摆地来到 1115 号的源码目录，东瞅瞅，西看看，挨个打开每一个文件，终于搞清楚了各个文件之间的关系。然后他按照一套很官方的格式，写了初步审核报告书，存了个文件名叫做 configure.scan 的文件，之后就回去睡觉去了。报告书的内容大约是这样：

```
#                                        -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.

AC_PREREQ([2.65])
AC_INIT([FULL-PACKAGE-NAME], [VERSION], [BUG-REPORT-ADDRESS])
AC_CONFIG_SRCDIR([board.h])
AC_CONFIG_HEADERS([config.h])

# Checks for programs.
AC_PROG_CC

# Checks for libraries.

# Checks for header files.
AC_CHECK_HEADERS([stdlib.h])

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.

AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

【修改报告】

懒蜗牛拿到报告书，当然知道，这只是万里长征才走完了第一步。赶紧叫来 gedit 小弟，修改报告书。他把一些完全不着边际的东西删掉，或者注释掉（也就是在那一行的前面加上#号）修改后的报告书是这个样子的：

```
#                                        -*- Autoconf -*-
# Process this file with Autoconf to produce a configure script.

#AC_PREREQ([2.64])
AC_INIT(main.c)
AM_INIT_AUTOMAKE(rubbish1115, 1.0)
#AC_CONFIG_SRCDIR([ai.c])
#AC_CONFIG_HEADERS([config.h])

# Checks for programs.
AC_PROG_CC

# Checks for libraries.

# Checks for header files.
#AC_CHECK_HEADERS([stdlib.h])

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.

AC_OUTPUT(Makefile)
```

看着挺多，其实真正有用的就下面这么几行。

- AC_INIT(main.c)——这一句说明这个工程的主要图纸是哪个文件。
- AM_INIT_AUTOMAKE(rubbish1115,1.0)——这一行是汇报这个项目的名称，叫做 rubbish1115，版本是 1.0 版。
- AC_PROG_CC——这一句是说，最终的 configure 需要检查 C 语言编译器是否正常。

- AC_OUTPUT(Makefile)——这一行是说明，最终的 configure 需要产生的文件，叫做 Makefile。

其他的，都是废话。

8.5.5 项目复审

懒蜗牛同学把改好的报告改名为 configure.in，然后就去叫 Aclocal 来看报告了。也就是运行了这个命令：

```
$aclocal
```

【Aclocal 的职能】

Aclocal 负责复查 Autoscan 的报告，并且根据里面的内容，做一些详细的注解和说明。并且把这些注解和说明也写成一个报告，叫做 aclocal.m4。

这里所谓的说明，主要是针对 configure.in 中的一些宏定义，进行详细的阐述。比如 configure.in 中写的“AC_PROG_CC”，只是说明了在最终的 configure 脚本中，要加入检查编译器的部分。但是编译器怎么个检查法？检查哪个编译器？都没说明白。这些在 Aclocal 的报告中都会有详细的解释——编译器要使用 GCC，configure 脚本中，要检测系统是否有 GCC 编译器。

随着懒蜗牛按下回车键，只见硬盘中的 Aclocal 不紧不慢地起床，伸着懒腰走进工作间，拿起桌上的茶水杯，掀开杯盖，撇一撇浮在水面上的茶叶，拿嘴吹一吹，喝一口，盖上杯盖，放下杯子，开始看报告。扫了两眼后，就知道是怎么回事了，也不用懒蜗牛多说话，直接写了一份复查报告，叫做 aclocal.m4，扔给了懒蜗牛，然后就赶紧回去继续睡觉去了。要说人家 Aclocal 的办事效率还真是不错，毕竟人家写的 aclocal.m4 不用懒蜗牛修改，直接就可以往上交了。

8.5.6 派遣分析师

主要领导终于出场了，懒蜗牛赶紧又叫来 Autoconf，让他指派分析师 configure。也就是运行了这个命令：

```
$autoconf
```

【Aotuconf 的职能】

Autoconf 就是专门负责指派分析师。他看了两份报告后，一般会沉思一会儿，说说目前如何困难，人手不足之类的话。最终在用户一再的苦苦哀求，以及威逼利诱之下，无可奈何地说：好，就给你派个分析师吧！如果顺利，一个 configure 脚本就诞生了。

不过懒蜗牛遇到的 Autoconf 这回倒是没太耽搁，看了 configure.in 和 aclocal.m4 两份报告后，很快就生成了 configure 脚本。由于懒蜗牛同学的这个程序很简单，因此 configure 脚本的内容也只有 4 千多行，不过麻雀虽小，五脏俱全，跟正规的 configure 脚本没有差别。

8.5.7 编写施工计划

那么有了 configure 脚本就完事了么？当然没有！都给你介绍了规划局有 4 个人，第 4

个还没出场呢，怎么能完呢？

这个 Autoconf 生成的 configure 要想去工作，是有条件的。他必须搭配规划局制定的施工计划——makefile.in 才能工作。有人会问，这个 makefile.in 是什么啊？我已经有了 Makefile 还要他干什么？咱不是说了么，你的那个 Makefile，甭管写得多么天花乱坠，也是白搭，人家规划局派出来的 configure 根本都不会瞧上一眼，人家 configure 要写自己的 Makefile 来用。你又得说了，那你这 configure 就赶快写出来自己的 Makefile 啊。你看你，不讲道理了不是，这 Makefile 文件那么复杂，哪能就这么凭空写出来，总得有个参考，有个蓝本，有个全市统一 Makefile 模板之类的东西吧。这个模板，就是 makefile.in。那么这个文件从哪来呢？这就用得着 Automake 了。

【Automake 的职能】

Automake 的职能就是专门写 configure 需要的 makefile.in（您看咱规划局给您搭配得多好）。不过 Automake 也不能直接就把 makefile.in 写出来。人家比较忙，这一点您也得理解。局里那么多重要的事情，今天学习，明天会餐，后天考察什么的。就算不会餐不考察，谁也免不了上个网、偷个菜、斗个地主扫个雷吧？

所以，Automake 是没工夫从头给你写出一份 makefile.in 的，你得先给 Automake 写好一个框架，然后人家才好动笔。这个框架，就叫做 Makefile.am。有了这个框架，交给 Automake，他就可以给你写出 makefile.in 了。

【编写 Makefile.am】

懒蜗牛同学没有打听好这个步骤，他直接找来 Automake，让他写 makefile.in。

Automake 拉着长声说："这个……我们这里呀，工作也比较忙嘛……要按说呢……这个文件我是应该给你写滴。不过我们这里每天这么多人来，我要是一个一个写，哪天才能写完呀。所以同志啊，为了帮助我们提高办事效率，也为了你自己早点拿到Makefile 的蓝本，更为了我们能够早日实现共同富裕奔小康——您是不是自己先写个草稿给我，我也好帮你赶快写出蓝本呀。"

懒蜗牛听得都快扔板砖了，心说不就你想犯懒这么点事么，至于跟我废这么多话么。赶紧动手写草稿吧，这个草稿叫做 Makefile.am。好在内容很简单：

```
AUTOMAKE_OPTIONS=foreign
bin_PROGRAMS=rubbish1115
rubbish1115_SOURCES=main.c ai.c board.c
```

- 第 1 行，是行业规定，一般都这么写。
- 第 2 行，说明编译之后的程序应该叫做 rubbish1115。
- 第 3 行，说这个工程包括 main.c、ai.c、board.c 这 3 个文件。

提示：第 1 行的 AUTOMAKE_OPTIONS=foreign 是 Automake 的选项。Automake 主要帮助开发 GNU 软件的人员来维护软件，所以在执行 Automake 时，默认会检查目录下是否存在标准 GNU 软件中应具备的文件，例如 NEWS、AUTHOR、ChangeLog 等文件。设置为 foreign 则 Automake 忽略掉对这些文件的检查。

就这么简单，草稿就写完了。之后再把 Automake 叫出来，总算是给写出了 Makefile 的蓝本——makefile.in。

做完了这些之后，这个工程就可以打包发布了。用户拿到这个包，解开之后，就直接

依次运行“./configure”、“make”、“make install”就把软件安装上了。

懒蜗牛欣喜地看着自己整出的这个像模像样的软件，看看 configure 脚本，4 千多行！心想：不知道的人看见这个脚本一定以为我是大牛吧，哈哈哈。再运行一下 configure，看看生成的 Makefile，500 多行，哈哈，俨然感觉自己已经成为高手了一样。于是，在懒蜗牛的 YY 中，我们结束了那一天的工作。

8.6 本章小结

咱们的懒蜗牛同学创造了不少的 rubbish 之后，算是对 Linux 下面的软件开发有了深入一些的了解了。什么编译原理、Makefile，还有怎么使用 configure，怎么使用 Autoconf，全都体验了一下。

从这以后，懒蜗牛算是更理解这笨兔子系统了。理解它的开源；理解它的简洁；理解它的效率；理解它的灵活。从此，懒蜗牛和笨兔子幸福地生活在了一起……